2006
Guide to the
CSTE COMMON BODY OF KNOWLEDGE

Software CERTIFICATIONS

Version 6.2

QUALITY ASSURANCE INSTITUTE WORLDWIDE

The Software Certification Program is Proudly Administered by the Quality Assurance Institute.

Table of Contents

Introduction to the CSTE Program Intro-1

Intro.1. Software Certification Overview Intro-2

Intro.1.1. Contact Us Intro-3

Intro.1.2. Program History Intro-3

Intro.1.3. Why Become Certified? Intro-3

Intro.1.4. Benefits of Becoming a CSTE Intro-3

Intro.1.4.1. Value Provided to the Profession Intro-4
Intro.1.4.2. Value Provided to the Individual Intro-4
Intro.1.4.3. Value Provided to the Employer Intro-5
Intro.1.4.4. How to Improve Testing Effectiveness Through CSTE Certification Intro-6

Intro.2. Meeting the CSTE Qualifications Intro-7

Intro.2.1. Prerequisites for Candidacy Intro-7

Intro.2.1.1. Educational and Professional Prerequisites Intro-7
Intro.2.1.2. Non-U.S. Prerequisites Intro-7
Intro.2.1.3. Expectations of the CSTE Intro-8

Intro.2.2. Code of Ethics Intro-9

Intro.2.2.1. Purpose Intro-9
Intro.2.2.2. Responsibility Intro-9
Intro.2.2.3. Professional Code of Conduct Intro-10
Intro.2.2.4. Grounds for Decertification Intro-11

Intro.2.3. Submitting the Initial Application Intro-11

Intro.2.3.1. Correcting Application Errors Intro-12
Intro.2.3.2. Submitting Application Changes Intro-13

Intro.2.4. Application-Examination Eligibility Requirements Intro-13

Intro.2.4.1. Filing a Retake Application Intro-13

Intro.3. Arranging to Sit and Take the Examination Intro-14

Intro.3.1. Scheduling to Take the Examination Intro-14

Intro.3.1.1. Rescheduling the Examination Sitting Intro-15

Intro.3.2. Receiving the Confirmation Letter Intro-15

Intro.3.3. Checking Examination Arrangements Intro-15

Intro.3.4. Arriving at the Examination Site Intro-16

Intro.3.4.1. No-shows Intro-16
Intro.4. How to Maintain Competency and Improve Value Intro-16
Intro.4.1. Continuing Professional Education Intro-16
Intro.4.2. Advanced CSTE Designations Intro-17
Intro.4.2.1. What is the Certification Competency Emphasis? Intro-17
Preparing for the CSTE Examination Intro-19
Intro.5. Assess Your CSTE 2006 CBOK Competency Intro-19
Intro.5.1. Complete the CSTE Skill Assessment Worksheet Intro-20
Intro.5.2. Calculate Your CSTE CBOK Competency Rating Intro-21
Intro.6. Understand the Key Principles Incorporated Into the Examination Intro-22
Intro.7. Review the List of References Intro-23
Intro.8. Initiate a Self-Study Program Intro-24
Intro.9. Take the Sample Examination Intro-24
CSTE 2006 Skill Assessment Worksheet SAW-1
. Assess Your Skills against the CSTE 2006 CBOK SAW-1
. Skill Category 1 – Software Testing Principles and Concepts SAW-2
. Skill Category 2 – Building the Test Environment SAW-3
. Skill Category 3 – Managing the Test Project SAW-4
. Skill Category 4 – Test Planning SAW-5
. Skill Category 5 – Executing the Test Plan SAW-6
. Skill Category 6 – Test Reporting Process SAW-7
. Skill Category 7 – User Acceptance Testing SAW-8
. Skill Category 8 – Testing Software Developed by Contractors SAW-9
. Skill Category 9 – Testing Internal Control SAW-10
. Skill Category 10 – Testing New Technologies SAW-11
. CSTE 2006 CBOK Competency Rating Table SAW-12

Skill Category 1

Software Testing Principles and Concepts 1-1

1.1. Vocabulary 1-1

1.1.1. Quality Assurance Versus Quality Control 1-2

1.1.1.1. Quality Assurance 1-2

1.1.1.2. Quality Control 1-3

1.1.2. The Cost of Quality 1-4

1.1.3. Software Quality Factors 1-5

1.1.3.1. How to Identify Important Software Quality Factors 1-6

1.1.3.2. Inventory Control System Example 1-10

1.1.4. How Quality is Defined 1-12

1.1.5. Definitions of Quality 1-13

1.1.6. What is Quality Software? 1-13

1.1.6.1. The Two Software Quality Gaps 1-13

1.1.6.2. What is Excellence? 1-14

1.2. What is Life Cycle Testing? 1-15

1.2.1. Why Do We Test Software? 1-15

1.2.2. Developers are not Good Testers 1-16

1.2.3. What is a Defect? 1-16

1.2.4. Software Process Defects 1-17

1.2.4.1. What Does It Mean For a Process To Be In or Out of Control? 1-17

1.2.4.2. Do Testers Need to Know SPC? 1-22

1.2.5. Software Product Defects 1-23

1.2.5.1. Software Design Defects 1-23

1.2.5.2. Data Defects 1-23

1.2.6. Finding Defects 1-24

1.3. Reducing the Frequency of Defects in Software Development 1-25

1.3.1. The Five Levels of Maturity 1-25

1.3.1.1. Level 1 – Ad Hoc 1-26

1.3.1.2. Level 2 – Control 1-27

1.3.1.3. Level 3 – Core Competency 1-27

1.3.1.4. Level 4 – Predictable 1-28

1.3.1.5. Level 5 – Innovative 1-28

1.3.2. Testers Need to Understand Process Maturity 1-28

1.4. Factors Affecting Software Testing 1-29

1.4.1. People Relationships 1-29

1.4.2. Scope of Testing 1-30

1.4.3. Misunderstanding Life Cycle Testing 1-31
1.4.3.1. Requirements 1-33
1.4.3.2. Design 1-33
1.4.3.3. Program (Build/Construction) 1-34
1.4.3.4. Test Process 1-34
1.4.3.5. Installation 1-34
1.4.3.6. Maintenance 1-34
1.4.4. Poorly Developed Test Planning 1-35
1.4.5. Testing Constraints 1-36
1.4.5.1. Budget and Schedule Constraints 1-36
1.4.5.2. Lacking or Poorly Written Requirements 1-38
1.4.5.3. Changes in Technology 1-39
1.4.5.4. Limited Tester Skills 1-40
1.5. Life Cycle Testing 1-40
1.6. Test Matrices 1-42
1.6.1. Cascading Test Matrices 1-44
1.7. Independent Testing 1-45
1.8. Tester's Workbench 1-47
1.8.1. What is a Process? 1-47
1.8.1.1. The PDCA View of a Process 1-48
1.8.1.2. The Workbench View of a Process 1-49
1.8.1.3. Workbenches are Incorporated into a Process 1-50
1.9. Levels of Testing 1-51
1.9.1. Verification versus Validation 1-52
1.9.1.1. Computer System Verification and Validation Examples 1-52
1.9.1.2. Functional and Structural Testing 1-55
1.9.2. Static versus Dynamic Testing 1-56
1.9.3. The "V" Concept of Testing 1-57
1.9.3.1. An 11-Step Software Testing Process Example 1-59
1.10. Testing Techniques 1-62
1.10.1. Structural versus Functional Technique Categories 1-62
1.10.1.1. Structural System Testing Technique Categories 1-63
1.10.1.2. Functional System Testing Technique Categories 1-71
1.10.2. Examples of Specific Testing Techniques 1-80
1.10.2.1. White-Box Testing 1-80
1.10.2.2. Black-Box Testing 1-81
1.10.2.3. Incremental Testing 1-81
1.10.2.4. Thread Testing 1-82
1.10.2.5. Requirements Tracing 1-83
1.10.2.6. Desk Checking and Peer Review 1-83
1.10.2.7. Walkthroughs, Inspections, and Reviews 1-84

1.10.2.8. Proof of Correctness Techniques 1-86
1.10.2.9. Simulation 1-87
1.10.2.10. Boundary Value Analysis 1-88
1.10.2.11. Error Guessing and Special Value Analysis 1-88
1.10.2.12. Cause-Effect Graphing 1-88
1.10.2.13. Design-Based Functional Testing 1-89
1.10.2.14. Coverage-Based Testing 1-89
1.10.2.15. Complexity-Based Testing 1-91
1.10.2.16. Statistical Analyses and Error Seeding 1-91
1.10.2.17. Mutation Analysis 1-91
1.10.2.18. Flow Analysis 1-92
1.10.2.19. Symbolic Execution 1-93
1.10.3. Combining Specific Testing Techniques 1-93

Skill Category 2

Building the Test Environment 2-1

2.1. Management Support 2-1
2.1.1. Management Tone 2-2
2.1.2. Integrity and Ethical Values 2-3
2.1.2.1. Incentives and Temptations 2-4
2.1.2.2. Providing and Communicating Moral Guidance 2-4
2.1.3. Commitment to Competence 2-5
2.1.4. Management's Philosophy and Operating Style 2-5
2.1.5. Organizational Structure 2-6
2.1.5.1. Assignment of Authority and Responsibility 2-6
2.1.5.2. Human Resource Policies and Practices 2-7
2.2. Test Work Processes 2-8
2.2.1. The Importance of Work Processes 2-8
2.2.2. Developing Work Processes 2-10
2.2.2.1. Defining the Attributes of a Standard for a Standard 2-10
2.2.2.2. Developing a Test Standard 2-10
2.2.3. Tester's Workbench 2-11
2.2.4. Responsibility for Building Work Processes 2-13
2.2.4.1. Responsibility for Policy 2-13
2.2.4.2. Responsibility for Standards and Procedures 2-14
2.2.4.3. Test Process Selection 2-15
2.2.4.4. Building a Process Engineering Organization 2-15
2.2.4.5. Professional Test Standards 2-20
2.2.5. Analysis and Improvement of the Test Process 2-21
2.2.5.1. Test Process Analysis 2-22
2.2.5.2. Continuous Improvement 2-26
2.2.5.3. Test Process Improvement Model 2-28
2.2.5.4. Test Process Alignment 2-32
2.2.5.5. Adapting the Test Process to Different Software Development Methodologies 2-33
2.3. Test Tools 2-36
2.3.1. Tool Development and Acquisition 2-36
2.3.1.1. Sequence of Events to Select Testing Tools 2-39
2.3.2. Classes of Test Tools 2-47
2.4. Testers Competency 2-49

Skill Category 3

Managing the Test Project 3-1

3.1. Test Administration 3-1

3.1.1. Test Planning 3-2

3.1.2. Budgeting 3-2

3.1.2.1. Budgeting Techniques 3-4

3.1.2.2. Tracking Budgeting Changes 3-5

3.1.3. Scheduling 3-6

3.1.4. Staffing 3-7

3.1.4.1. Test Team Approaches 3-8

3.1.5. Customization of the Test Process 3-10

3.2. Test Supervision 3-11

3.2.1. Communication Skills 3-11

3.2.1.1. Written and Oral Communication 3-12

3.2.1.2. Listening Skills 3-15

3.2.1.3. The 3-Step Listening Process 3-16

3.2.1.4. Interviewing Skills 3-18

3.2.1.5. Analyzing Skills 3-19

3.2.2. Negotiation and Complaint Resolution Skills 3-21

3.2.2.1. Negotiation 3-21

3.2.2.2. Resolving Complaints 3-23

3.2.2.3. The 4-Step Complaint-Resolution Process 3-23

3.2.3. Judgment 3-25

3.2.4. Providing Constructive Criticism 3-25

3.2.5. Project Relationships 3-26

3.2.6. Motivation, Mentoring, and Recognition 3-28

3.2.6.1. Motivation 3-28

3.2.6.2. Mentoring 3-28

3.2.6.3. Recognition 3-29

3.3. Test Leadership 3-29

3.3.1. Chairing Meetings 3-29

3.3.2. Team Building 3-30

3.3.2.1. Team Development 3-30

3.3.2.2. Team Member Interaction 3-31

3.3.2.3. Team Ethics 3-32

3.3.2.4. Team Rewards 3-33

3.3.3. Quality Management Organizational Structure 3-33

3.3.4. Code of Ethics 3-35
3.3.4.1. Responsibility 3-36
3.4. Managing Change 3-36
3.4.1. Software Configuration Management 3-36
3.4.2. Software Change Management 3-37
3.4.3. Software Version Control 3-38
3.4.3.1. Example 3-38

Skill Category 4

Test Planning 4-1

4.1. Risk Concepts and Vocabulary 4-1
4.2. Risks Associated with Software Development 4-4
4.2.1. Improper Use of Technology 4-4
4.2.2. Repetition of Errors 4-5
4.2.3. Cascading of Errors 4-5
4.2.4. Illogical Processing 4-6
4.2.5. Inability to Translate User Needs into Technical Requirements . . .4-6
4.2.6. Inability to Control Technology 4-7
4.2.7. Incorrect Entry of Data 4-7
4.2.8. Concentration of Data 4-8
4.2.9. Inability to React Quickly 4-8
4.2.10. Inability to Substantiate Processing 4-9
4.2.11. Concentration of Responsibilities 4-9
4.2.12. Erroneous or Falsified Input Data 4-10
4.2.13. Misuse by Authorized End Users 4-11
4.2.14. Uncontrolled System Access 4-11
4.2.15. Ineffective Security and Privacy Practices for the Application . .4-12
4.2.16. Procedural Errors during Operations 4-12
4.2.16.1. Procedures and Controls 4-12
4.2.16.2. Storage Media Handling 4-13
4.2.17. Program Errors 4-13
4.2.18. Operating System Flaws 4-14
4.2.19. Communications System Failure 4-15
4.2.19.1. Accidental Failures 4-15
4.2.19.2. Intentional Acts 4-15
4.3. Risks Associated with Software Testing 4-16
4.3.1. Premature Release Risk 4-18
4.4. Risk Analysis 4-19
4.4.1. Risk Analysis Process 4-20
4.4.1.1. Form the Risk Analysis Team 4-20
4.4.1.2. Identify Risks 4-20
4.4.1.3. Estimate the Magnitude of the Risk 4-21

4.4.1.4. Select Testing Priorities 4-24
4.5. Risk Management 4-24
4.5.1. Risk Reduction Methods 4-25
4.5.2. Contingency Planning 4-26
4.6. Prerequisites to Test Planning 4-27
4.6.1. Test Objectives 4-27
4.6.2. Acceptance Criteria 4-27
4.6.3. Assumptions 4-28
4.6.4. People Issues 4-28
4.6.5. Constraints 4-29
4.7. Create the Test Plan 4-29
4.7.1. Understand the Characteristics of the Software being Developed . 4-31
4.7.2. Build the Test Plan 4-32
4.7.2.1. Set Test Objectives 4-32
4.7.2.2. Develop the Test Matrix 4-33
4.7.2.3. Define Test Administration 4-37
4.7.2.4. State Test Plan General Information 4-39
4.7.3. Write the Test Plan 4-40
4.7.3.1. Guidelines to Writing the Test Plan 4-41
4.7.3.2. Test Plan Standard 4-41

Skill Category 5

Executing the Test Plan 5-1

5.1. Test Case Design 5-1

5.1.1. Functional Test Cases 5-2

5.1.1.1. Design Specific Tests for Testing Code 5-2
5.1.1.2. Functional Testing Independent of the Specification Technique 5-2
5.1.1.3. Functional Testing Based on the Interface 5-2
5.1.1.4. Functional Testing Based on the Function to be Computed 5-3
5.1.1.5. Functional Testing Dependent on the Specification Technique 5-4

5.1.2. Structural Test Cases 5-5

5.1.2.1. Structural Analysis 5-5
5.1.2.2. Structural Testing 5-5

5.1.3. Erroneous Test Cases 5-7

5.1.3.1. Statistical Methods 5-7
5.1.3.2. Error-Based Testing 5-8

5.1.4. Stress Test Cases 5-10

5.1.5. Test Scripts 5-11

5.1.5.1. Determine Testing Levels 5-11
5.1.5.2. Develop the Scripts 5-12
5.1.5.3. Execute the Script 5-14
5.1.5.4. Analyze the Results 5-14
5.1.5.5. Maintain Scripts 5-15

5.1.6. Use Cases 5-17

5.1.6.1. Build a System Boundary Diagram 5-18
5.1.6.2. Define Use Cases 5-19
5.1.6.3. Develop Test Cases 5-20
5.1.6.4. Test Objective 5-21

5.1.7. Building Test Cases 5-21

5.1.8. Process for Building Test Cases 5-22

5.1.9. Example of Creating Test Cases for a Payroll Application 5-23

5.2. Test Coverage 5-26

5.3. Performing Tests 5-27

5.3.1. Platforms 5-27

5.3.2. Test Cycle Strategy 5-27

5.3.3. Use of Tools in Testing 5-28

5.3.3.1. Test Documentation 5-28
5.3.3.2. Test Drivers 5-28
5.3.3.3. Automatic Test Systems and Test Languages 5-29

5.3.4. Perform Tests 5-29
5.3.4.1. Perform Unit Testing 5-30
5.3.4.2. Perform Integration Test 5-30
5.3.4.3. Perform System Test 5-31
5.3.5. When is Testing Complete? 5-32
5.3.6. General Concerns 5-32
5.4. Recording Test Results 5-32
5.4.1. Problem Deviation 5-33
5.4.2. Problem Effect 5-35
5.4.3. Problem Cause 5-36
5.4.4. Use of Test Results 5-36
5.5. Defect Management 5-37
5.5.1. Defect Naming Guidelines 5-38
5.5.1.1. Name of the Defect 5-38
5.5.1.2. Defect Severity 5-38
5.5.1.3. Defect Type 5-38
5.5.1.4. Defect Class 5-38
5.5.2. The Defect Management Process 5-39
5.5.2.1. Defect Prevention 5-39
5.5.2.2. Deliverable Baseline 5-43
5.5.2.3. Defect Discovery 5-45
5.5.2.4. Defect Resolution 5-50
5.5.2.5. Process Improvement 5-51

Skill Category 6

Test Reporting Process 6-1

6.1. Prerequisites to Test Reporting 6-1

6.1.1. Define and Collect Test Status Data 6-2

6.1.1.1. Test Results Data 6-2
6.1.1.2. Test Case Results and Test Verification Results 6-2
6.1.1.3. Defects 6-3
6.1.1.4. Efficiency 6-3

6.1.2. Define Test Metrics used in Reporting 6-3

6.1.3. Define Effective Test Metrics 6-5

6.1.3.1. Objective versus Subjective Measures 6-6
6.1.3.2. How Do You Know a Metric is Good? 6-7
6.1.3.3. Standard Units of Measure 6-7
6.1.3.4. Productivity versus Quality 6-8
6.1.3.5. Test Metric Categories 6-8

6.2. Test Tools used to Build Test Reports 6-11

6.2.1. Pareto Charts 6-11

6.2.1.1. Deployment 6-11
6.2.1.2. Examples 6-13
6.2.1.3. Results 6-13
6.2.1.4. Recommendations 6-14

6.2.2. Pareto Voting 6-14

6.2.2.1. Deployment 6-14
6.2.2.2. Example 6-15

6.2.3. Cause and Effect Diagrams 6-15

6.2.3.1. Deployment 6-15
6.2.3.2. Results 6-17
6.2.3.3. Examples 6-17
6.2.3.4. Recommendation 6-18

6.2.4. Check Sheets 6-18

6.2.4.1. Deployment 6-18
6.2.4.2. Results 6-18
6.2.4.3. Examples 6-19
6.2.4.4. Recommendations 6-19
6.2.4.5. Example Check Sheet 6-20

6.2.5. Histograms 6-20

6.2.5.1. Variation of a Histogram 6-21
6.2.5.2. Deployment 6-21
6.2.5.3. Results 6-22
6.2.5.4. Examples 6-22

6.2.5.5. Recommendations 6-22
6.2.6. Run Charts 6-22
6.2.6.1. Deployment 6-23
6.2.6.2. Results 6-23
6.2.6.3. Examples 6-24
6.2.6.4. Recommendations 6-24
6.2.7. Scatter Plot Diagrams 6-24
6.2.7.1. Deployment 6-24
6.2.7.2. Results 6-27
6.2.7.3. Examples 6-27
6.2.8. Regression Analysis 6-28
6.2.8.1. Deployment 6-28
6.2.8.2. Results 6-29
6.2.9. Multivariate Analysis 6-29
6.2.9.1. Deployment 6-30
6.2.9.2. Results 6-30
6.2.10. Control Charts 6-30
6.2.10.1. Deployment 6-31
6.2.10.2. Results 6-32
6.2.10.3. Examples 6-32
6.3. Test Tools used to Enhance Test Reporting 6-32
6.3.1. Benchmarking 6-32
6.3.1.1. A Ten-Step Process to Collect Benchmark Data 6-33
6.3.2. Quality Function Deployment 6-36
6.4. Reporting Test Results 6-37
6.4.1. Current Status Test Reports 6-38
6.4.1.1. Function Test Matrix 6-39
6.4.1.2. Defect Status Report 6-40
6.4.1.3. Functional Testing Status Report 6-41
6.4.1.4. Functions Working Timeline 6-42
6.4.1.5. Expected versus Actual Defects Uncovered Timeline 6-43
6.4.1.6. Defects Uncovered versus Corrected Gap Timeline 6-44
6.4.1.7. Average Age of Uncorrected Defects by Type 6-45
6.4.1.8. Defect Distribution Report 6-46
6.4.1.9. Relative Defect Distribution Report 6-47
6.4.1.10. Testing Action Report 6-48
6.4.1.11. Individual Project Component Test Results 6-49
6.4.1.12. Summary Project Status Report 6-51
6.4.1.13. Individual Project Status Report 6-52
6.4.2. Final Test Reports 6-55
6.4.2.1. Description of Test Reports 6-56
6.4.2.2. Integration Test Report 6-57

6.4.2.3. System Test Report 6-58
6.4.3. Guidelines for Report Writing 6-59

Skill Category 7

User Acceptance Testing 7-1

7.1. Acceptance Testing Concepts 7-1
7.1.1. Difference between Acceptance Test and System Test 7-4
7.2. Roles and Responsibilities 7-5
7.2.1. User's Role 7-6
7.2.2. Software Tester's Role 7-6
7.3. Acceptance Test Planning 7-7
7.3.1. Acceptance Criteria 7-7
7.3.2. Acceptance Test Plan 7-10
7.3.3. Use Case Test Data 7-11
7.4. Acceptance Test Execution 7-12
7.4.1. Execute the Acceptance Test Plan 7-12
7.4.2. Acceptance Decision 7-13

Skill Category 8

Testing Software Developed by Contractors 8-1

8.1. Challenges in Testing Acquired Software 8-2

8.1.1. Purchased COTS software 8-2

8.1.1.1. Evaluation versus Assessment 8-3

8.1.2. Contracted Software 8-3

8.1.2.1. Additional Differences with Contractors in another Country (Offshore) . . 8-3

8.1.2.2. Software Tester's Responsibility for Software Developed by a Contractor 8-4

8.2. COTS Software Test Process 8-5

8.2.1. Assure Completeness of Needs Specification 8-5

8.2.1.1. Define Critical Success Factor 8-6

8.2.1.2. Determine Compatibility with Your Computer Environment 8-7

8.2.1.3. Assure the Software can be Integrated into Your Business System Work Flow 8-8

8.2.1.4. Demonstrate the Software in Operation 8-9

8.2.1.5. Evaluate the People Fit 8-11

8.2.1.6. Acceptance Test the COTS Software 8-11

8.3. Contracted Software Test Process 8-11

8.3.1. Assure the Process for Contracting Software is Adequate 8-12

8.3.2. Review the Adequacy of the Contractor's Test Plan 8-17

8.3.3. Assure Development is Effective and Efficient 8-18

8.3.4. Perform Acceptance Testing on the Software 8-18

8.3.5. Issue a Report on the Adequacy of the Software to Meet the Needs of the Organization 8-19

8.3.6. Ensure Knowledge Transfer Occurs and Intellectual Property Rights are Protected 8-19

8.3.7. Incorporate Copyrighted Material into the Contractor's Manuals .8-20

8.3.8. Assure the Ongoing Operation and Maintenance of the Contracted Software 8-20

8.3.9. Assure the Effectiveness of Contractual Relations 8-22

Skill Category 9
Testing Software Controls and the Adequacy of Security Procedures 9-1

9.1. Principles and Concepts of Internal Control 9-2
9.1.1. Internal Control Responsibilities 9-3
9.1.2. Software Tester's Internal Control Responsibilities 9-3
9.1.3. Internal Auditor's Internal Control Responsibilities 9-3
9.1.4. Risk versus Control 9-5
9.1.5. Environmental versus Transaction Processing Controls 9-5
9.1.5.1. Environmental or General Controls 9-5
9.1.6. Transaction Processing Controls 9-6
9.1.7. Preventive, Detective and Corrective Controls 9-8
9.1.7.1. Preventive Controls 9-8
9.1.7.2. Detective Controls 9-13
9.2. Internal Control Models 9-17
9.2.1. COSO Enterprise Risk Management (ERM) Model 9-17
9.2.1.1. The ERM Process 9-17
9.2.1.2. Components of ERM 9-17
9.2.2. COSO Internal Control Framework Model 9-19
9.2.2.1. Example of a Transaction Processing Internal Control System 9-21
9.2.3. CobiT Model 9-22
9.3. Testing Internal Controls 9-23
9.3.1. Perform Risk Assessment 9-24
9.3.2. Test Transaction Processing Controls 9-25
9.3.2.1. Transaction Origination 9-26
9.3.2.2. Transaction Entry 9-26
9.3.2.3. Transaction Communications 9-27
9.3.2.4. Transaction Processing 9-27
9.3.2.5. Database Storage and Retrieval 9-27
9.3.2.6. Transaction Output 9-27
9.4. Testing Security Controls 9-27
9.4.1. Task 1 –Where Security is Vulnerable to Penetration 9-28
9.4.1.1. Accidental versus Intentional Losses 9-29
9.4.2. Task 2 – Building a Penetration Point Matrix 9-30
9.4.2.1. Controlling People by Controlling Activities 9-30
9.4.2.2. Selecting Computer Security Activities 9-31

9.4.2.3. Controlling Business Transactions 9-35
9.4.2.4. Characteristics of Security Penetration 9-37
9.4.3. Task 3 – Assess Security Awareness Training 9-43
9.4.3.1. Step 1 – Create a Security Awareness Policy 9-44
9.4.3.2. Step 2 – Develop a Security Awareness Strategy 9-45
9.4.3.3. Step 3 – Assign the Roles for Security Awareness 9-48
9.4.4. Task 4 – Understand the Attributes of an Effective Security Control 9-50
9.4.5. Task 5 – Selecting Techniques to Test Security 9-51
9.4.5.1. Step 1 – Understand Security Testing Techniques 9-52
9.4.5.2. Step 2 – Select Security Testing Techniques Based on the Strengths and Weaknesses of Those Techniques 9-52
9.4.5.3. Step 3 – Determine the Frequency of Use of Security Testing Techniques Based on the System Category 9-54

Skill Category 10

Testing New Technologies 10-1

10.1. Risks Associated with New Technology 10-1

10.2. Newer IT Technologies that Impact Software Testing 10-3

10.2.1. Web-Based Applications 10-3

10.2.2. Distributed Application Architecture 10-4

10.2.2.1. Traditional Client-Server Systems 10-4

10.2.2.2. Thin- versus Thick-Client Systems 10-5

10.2.3. Wireless Technologies 10-5

10.2.3.1. Important Issues for Wireless 10-6

10.2.4. New Application Business Models 10-7

10.2.4.1. e-Commerce 10-7

10.2.4.2. e-Business 10-7

10.2.5. New Communication Methods 10-8

10.2.5.1. Wireless Applications 10-8

10.2.6. New Testing Tools 10-11

10.2.6.1. Test Automation 10-11

10.3. Testing the Effectiveness of Integrating New Technology 10-12

10.3.1. Determine the Process Maturity Level of the Technology 10-13

10.3.1.1. Level 1 – People-Dependent Technology 10-14

10.3.1.2. Level 2 – Use Description-Dependent Technology Processes 10-14

10.3.1.3. Level 3 – Use of Technology 10-14

10.3.1.4. Level 4 – Quantitatively Measured Technology 10-14

10.3.1.5. Level 5 – Optimized Use of Technology 10-14

10.3.2. Test the Controls over Implementing the New Technology . . . 10-15

10.3.2.1. Test Actual Performance versus Stated Performance 10-15

10.3.2.2. Test the Adequacy of the Current Processes to Control the Technology 10-16

10.3.3. Test the Adequacy of Staff Skills to Use the Technology 10-17

Appendix A

Vocabulary A-1

Appendix B

References B-1

How to Take the CSTE Examination C-1

C.1. CSTE Examination Overview C-1

C.1.1. Software Testing Theory C-2

C.1.2. Software Testing Practice C-2

C.2. Guidelines to Answer Questions C-2

C.3. Sample CSTE Examination C-5

C.3.1. Part 1 and Part 3 Multiple-Choice Questions C-5

C.3.2. Part 1 and Part 3 Multiple-Choice Answers C-11

C.3.3. Part 2 and Part 4 Essay Questions and Answers C-12

C.3.3.1. Part 2 – Software Testing Theory Essay Questions C-12

C.3.3.2. Part 2 – Software Testing Theory Essay Answers C-15

C.3.3.3. Part 4 – Software Testing Practice Essay Questions C-19

C.3.3.4. Part 4 – Quality Assurance Practice Essay Answers C-22

This page intentionally left blank.

Introduction to the CSTE Program

The Certified Software Tester (CSTE) program was developed by leading software testing professionals as a means of recognizing software testers who demonstrate a predefined level of testing competency. The CSTE program is directed by an independent Certification Board and administered by the Quality Assurance Institute (QAI). The program was developed to provide value to the profession, the individual, the employer, and co-workers.

The CSTE certification entails an aggressive educational program that tests the level of competence in the principles and practices of testing and control in the Information Technology (IT) profession. These principles and practices are defined by the Certification Board as the Common Body of Knowledge (CBOK). The Certification Board will periodically update the CBOK to reflect changing software testing and control, as well as changes in computer technology. These updates should occur approximately every three years.

Software Certification Overview	**Intro-2**
Meeting the CSTE Qualifications	**Intro-7**
Arranging to Sit and Take the Examination	**Intro-14**
How to Maintain Competency and Improve Value	**Intro-16**

Be sure to check the Software Certifications Web site for up-to-date information on the CSTE program, examination sites and schedules, and What's New at:

www.softwarecertifications.org

Using this product does not constitute, nor imply, the successful passing of the CSTE certification examination.

Intro.1 Software Certification Overview

Software Certifications is recognized worldwide as the standard for IT testing professionals. Certification is a big step; a big decision. Certification identifies an individual as a test leader and earns the candidate the respect of colleagues and managers. It is formal acknowledgement that the IT recipient has an overall understanding of the disciplines and skills represented in a comprehensive Common Body of Knowledge (CBOK) for a respective software discipline.

The CSTE program demonstrates the following objectives to establish standards for initial qualification and continuing improvement of professional competence. This certification program helps to:

1. Define the tasks (skill categories) associated with software testing duties in order to evaluate skill mastery.
2. Demonstrate an individual's willingness to improve professionally.
3. Acknowledge attainment of an acceptable standard of professional competency.
4. Aid organizations in selecting and promoting qualified individuals.
5. Motivate personnel having software testing responsibilities to maintain their professional competency.
6. Assist individuals in improving and enhancing their organization's software testing programs (i.e., provide a mechanism to lead a professional).

In addition to CSTE, Software Certifications also offer the following software certifications. See "How to Maintain Competency and Improve Value" on page 16 for more information on the certifications for advanced and master levels.

- Software Testers
  - Advanced Software Tester (ASTE)
  - Master Software Tester (MSTE)
- Software Quality Analysts
  - Certified Software Quality Analyst (CSQA)
  - Advanced Software Quality Analyst (ASQA)
  - Master Software Quality Analyst (MSQA)
- Software Project Manager
  - Certified Software Project Manager (CSPM)

One or more of these certifications is frequently a prerequisite for promotion or acquiring a new position. See www.qaiworldwide.org and www.softwarecertifications.org for detailed information on all software certifications available including:

- Preparation Courses
- Examination Schedules
- Conferences and Seminars
- In-house Training Courses

Intro.1.1 Contact Us

Software Certifications
Phone: (407)-472-8100
Fax: (407)-398-6817

CSTE questions? E-mail: certify@softwarecertifications.org

Intro.1.2 Program History

QAI was established in 1980 as a professional association formed to represent the software testing industry. The first certification began development in 1985 and the first formal examination process was launched in 1990. Today, Software Certifications, administered by QAI, is global. Since its inception, Software Certifications has certified over 27,000 IT professionals in Australia, Barbados, Belgium, Bermuda, Brazil, Canada, China, Egypt, Hong Kong, India, Israel, Korea, Mexico, New Zealand, Puerto Rico, Saudi Arabia, Singapore, South Africa, United Kingdom, United Arab Emirates, and the United States.

Intro.1.3 Why Become Certified?

As the IT industry becomes more competitive, management must be able to distinguish professional and skilled individuals in the field when hiring. Certification demonstrates a level of understanding in carrying out software testing principles and practices that management can depend upon.

Acquiring the designation of CSTE indicates a professional level of competence in software testing. CSTEs become members of a recognized professional group and receive recognition for their competence by businesses and professional associates, potentially more rapid career advancement, and greater acceptance in the role as advisor to management.

Intro.1.4 Benefits of Becoming a CSTE

As stated above, the CSTE program was developed to provide value to the profession, the individual, the employer, and co-workers. The following information is data collected from CSTEs in the IT industry – a real testimonial to the benefits and reasons to make the effort to become a CSTE.

Intro.1.4.1 Value Provided to the Profession

Software testing is often viewed as a software project task, even though many individuals are full-time testing professionals. The CSTE program was designed to recognize software testing professionals by providing:

- Common Body of Knowledge (CBOK)

The Certification Board defines the skills upon which the software testing certification is based. The current CBOK includes 10 skill categories fully described in this preparation guide – see Skill Category 1 through Skill Category 10.

- Examination Process to Evaluate Competency

The successful candidate must pass a four-part examination that is based on the CBOK. You must receive a grade of 75%, averaged over all four parts of the examination, to become certified. See "How to Take the CSTE Examination" for a sample examination and answers to help you prepare for the actual examination.

- Code of Ethics

The successful candidate must agree to abide by a professional Code of Ethics as specified by the Certification Board. See "Code of Ethics" on page 9 for an explanation of the ethical behaviors expected of all certified professionals.

Intro.1.4.2 Value Provided to the Individual

The individual obtaining the CSTE certification receives the following values:

- Recognition by Peers of Personal Desire to Improve

Approximately eighty percent (80%) of all CSTEs stated that a personal desire for self-improvement and peer recognition was the main reason for obtaining the CSTE certification. Fifteen percent (15%) were required by their employer to sit for the examination, and 10% were preparing themselves for an improved testing-related position.

Many CSTEs indicated that while their employer did not require CSTE certification, it was strongly encouraged.

- Increased Confidence in Personal Capabilities

Eighty-five percent (85%) of the CSTEs stated that passing the examination increased their confidence to perform their job more effectively. Much of that confidence came from studying for the examination.

- Recognition by IT Management for Professional Achievement

Most CSTEs stated that their management greatly respects those who put forth the personal effort needed for self-improvement. IT organizations recognized and rewarded individuals in the following ways:

- Thirteen percent (13%) received an immediate average one-time bonus of $610, with a range of $250 to $2,500.
- Twelve percent (12%) received an immediate average salary increase of 10%, with a range of 2% to 50%.

Non-monetary recognitions were:

- Thirty-six percent (36%) were recognized in staff meetings.
- Twenty percent (20%) in newsletters or e-mail.
- Many received rewards, management visits or calls, and lunch with the boss.

Within the first 18 months after receipt of the CSTE certification, of the successful candidates:

- Twenty-seven percent (27%) received an average salary increase of 23%, with a range of 2% to 100%.
- Twenty-three percent (23%) were promoted, 25% received a better assignment and 13% a new assignment.

Intro.1.4.3 Value Provided to the Employer

With the need for increased software testing and reliability, employing CSTEs provides value in these ways:

Intro.1.4.3.1 Increased Confidence by IT Users and Customers

IT users and customers expressed confidence in IT to effectively build or acquire software when certified testing practitioners were involved.

Intro.1.4.3.2 Improved Processes to Build/Acquire/Maintain, Operate and Measure Software

CSTEs use their knowledge and skills to continuously improve the IT work processes. CSTEs know what to measure, how to measure it, and then prepare an analysis to aid in the decision-making process.

Intro.1.4.3.3 Independent Assessment of Testing Competencies

The CSTE program is directed by a Certification Board of independent testing experts. Through examination and recertification, they provide an independent assessment of the CSTE's testing competencies, based on a continuously strengthening Common Body of Knowledge for testing practitioners.

Intro.1.4.3.4 Testing Competencies Maintained Through Recertification

Yesterday's testing competencies are inadequate for today's challenges. CSTE recertification is a process that helps assure the CSTE's skills remain current. The recertification process requires CSTEs to obtain 40 hours of testing-related training per year in topics specified by the Certification Board.

From an IT director's perspective, this is employee-initiated testing training. Most, if not all CSTEs, do this training during their personal time. IT organizations gain three benefits from CSTE recertification: 1) employees initiate improvement; 2) testing practitioners obtain competencies in testing methods and techniques; and 3) employees train during personal time.

Intro.1.4.3.5 Value Provided to Co-Workers

The drive for self-improvement is a special trait that manifests itself in providing these values to co-workers:

Intro.1.4.3.6 Mentoring the Testing Staff

Forty-five percent (45%) of the CSTEs mentor their testing colleagues by conducting training classes; encouraging staff to become certified; and acting as a resource to the staff on sources of IT testing-related information.

Intro.1.4.3.7 Testing Resource to "IT" Staff

CSTEs are recognized as experts in testing and are used heavily for advice, counseling, and for recommendations on software construction and testing.

Intro.1.4.3.8 Role Model for Testing Practitioners

CSTEs are the IT role models for individuals with testing responsibilities to become more effective in performing their job responsibilities.

Intro.1.4.4 How to Improve Testing Effectiveness Through CSTE Certification

A "driver" for improved IT effectiveness is the integration of the CSTE certification program in your "IT" career development plan. This can be accomplished by:

- Creating an awareness of the CSTE Program and its benefits to your testing practitioners.
- Requiring or encouraging your testing practitioners to become certified.
- Recognizing and rewarding successful candidates.
- Supporting recertification as a means of maintaining testing competency.

QAI, as CSTE program administrators, will assist you in this effort.

See www.qaiworldwide.org for detailed information.

Intro.2 Meeting the CSTE Qualifications

To become certified as a Certified Software Tester, every candidate must first meet these qualifications:

1. Satisfy all of the prerequisites required prior to applying for candidacy – educational and professional prerequisites including non-U.S. prerequisites, recommendations for preparing for the examination, and understanding what will be expected once you are a CSTE.
2. Subscribe to the Code of Ethics as described on page Intro-9.
3. Submit a completed Certification Candidacy Application. See "Submitting the Initial Application" on page Intro-11 for information on all the materials needed to submit your application.

Intro.2.1 Prerequisites for Candidacy

Before you submit your application, first check that you satisfy the educational and professional prerequisites described below and understand what is expected of the CSTE after certification.

Intro.2.1.1 Educational and Professional Prerequisites

To qualify for candidacy, each applicant must meet one of three credentials:

1. A bachelor's degree from an accredited college-level institution.
2. An associate's degree and two years of experience in the information services field.

OR

3. Six years of experience in the information services field.

Intro.2.1.2 Non-U.S. Prerequisites

Educational requirements for Software Certifications are stated following the terms, customs, and requirements typically encountered in the United States. However, authority has been given to specific organizations sponsoring the examination process outside the United States to examine and modify educational and experience criteria within their countries. Each country's criteria will be based on the following framework:

- Candidates should possess qualifications equal to other professionals of similar status.
- Candidates should possess the superior knowledge and skills needed to carry out all designated responsibilities in a preeminent manner.
- Candidates' education and experience must be broadly based to meet a wide range of responsibilities and expectations.

- Successful candidates must be able to execute suitable testing principles and practices in an array of diverse assignments and clearly communicate appropriate conclusions and recommendations.

Note: When submitting academic qualifications, the candidate must ensure that the materials are in sufficient detail so that the Software Certifications Board can determine equivalency. The Board is the final judge of acceptability of any alternative educational or experience-based criteria submitted by any applicant.

Intro.2.1.3 Expectations of the CSTE

Knowledge within a profession doesn't stand still. Having passed the CSTE examination, a certificant has demonstrated knowledge of the designation's CBOK at the point in time of the examination. In order to stay current in the field, as knowledge and techniques mature, the certificant must be actively engaged in professional practice, and seek opportunities to stay aware of, and learn, emerging practices.

The CSTE is required to submit 120 credit hours of Continuing Professional Education (CPE) every three years to maintain certification or take an examination for recertification. Any special exceptions to the CPE requirements are to be directed to Software Certifications. Certified professionals are generally expected to:

- Attend professional conferences to stay aware of activities and trends in the profession.
- Take education and training courses to continually update skills and competencies.
- Develop and offer training to share knowledge and skills with other professionals and the public.
- Publish information in order to disseminate personal, project, and research experiences.
- Participate in the profession through active committee memberships and formal special interest groups.

The CSTE is expected not only to possess the skills required to pass the CSTE examination but also to be a change agent: someone who can change the culture and work habits of individuals (or someone who can act in an advisory position to upper management) to make quality in software testing happen.

Intro.2.1.3.1 Professional Skill Proficiency Responsibilities

In preparing yourself for the profession of IT software testing and to become more effective in your current job, you need to become aware of the three C's of today's workplace:

- *Change* – The speed of change in technology and in the way work is performed is accelerating. Without continuous skill improvement, you will become obsolete in the marketplace.
- *Complexity* – Information technology is becoming more complex, not less complex. Thus, achieving quality, with regard to software testing in the information technology

environment, will become more complex. You must update your skill proficiency in order to deal with this increased complexity.

- *Competition* – The ability to demonstrate mastery of multiple skills makes you a more desirable candidate for any professional position. While hard work does not guarantee you success, few, if any, achieve success without hard work. CSTE certification is one form of achievement. CSTE certification is proof that you've mastered a basic skill set recognized worldwide in the information technology arena.

Intro.2.1.3.2 Develop a Lifetime Learning Habit

Become a lifelong learner in order to perform your current job effectively and remain marketable in an era of the three C's. You cannot rely on your current knowledge to meet tomorrow's job demands. The responsibility for success lies within your own control.

Perhaps the most important single thing you can do to improve yourself professionally and personally is to develop a lifetime learning habit.

REMEMBER: "If it is going to be—it's up to me."

Intro.2.2 Code of Ethics

An applicant for certification must subscribe to the following Code of Ethics that outlines the ethical behaviors expected of all certified professionals. Software Certifications includes processes and procedures for monitoring certificant's adherence to these policies. Failure to adhere to the requirements of the Code is grounds for decertification of the individual by the Software Certifications Board.

Intro.2.2.1 Purpose

A distinguishing mark of a profession is acceptance by its members of responsibility to the interests of those it serves. Those certified must maintain high standards of conduct in order to effectively discharge their responsibility.

Intro.2.2.2 Responsibility

This Code of Ethics is applicable to all certified by Software Certifications. Acceptance of any certification designation is a voluntary action. By acceptance, those certified assume an obligation of self-discipline beyond the requirements of laws and regulations.

The standards of conduct set forth in this Code of Ethics provide basic principles in the practice of information services testing. Those certified should realize that their individual judgment is required in the application of these principles.

Those certified shall use their respective designations with discretion and in a dignified manner, fully aware of what the designation denotes. The designation shall also be used in a manner consistent with all statutory requirements.

Those certified who are judged by the Software Certifications Board to be in violation of the standards of conduct of the Code of Ethics shall be subject to forfeiture of their designation.

Intro.2.2.3 Professional Code of Conduct

Software Certifications certificate holders shall:

1. Exercise honesty, objectivity, and diligence in the performance of their duties and responsibilities.
2. Exhibit loyalty in all matters pertaining to the affairs of their organization or to whomever they may be rendering a service. However, they shall not knowingly be party to any illegal or improper activity.
3. Not engage in acts or activities that are discreditable to the profession of information services testing or their organization.
4. Refrain from entering any activity that may be in conflict with the interest of their organization or would prejudice their ability to carry out objectively their duties and responsibilities.
5. Not accept anything of value from an employee, client, customer, supplier, or business associate of their organization that would impair, or be presumed to impair, their professional judgment and integrity.
6. Undertake only those services that they can reasonably expect to complete with professional competence.
7. Be prudent in the use of information acquired in the course of their duties. They shall not use confidential information for any personal gain nor in any manner that would be contrary to law or detrimental to the welfare of their organization.
8. Reveal all material facts known to them that, if not revealed, could either distort reports of operation under review or conceal unlawful practices.
9. Continually strive for improvement in their proficiency, and in the effectiveness and quality of their service.
10. In the practice of their profession, shall be ever mindful of their obligation to maintain the high standards of competence, morality, and dignity promulgated by this Code of Ethics.
11. Maintain and improve their professional competency through continuing education.
12. Cooperate in the development and interchange of knowledge for mutual professional benefit.

13. Maintain high personal standards of moral responsibility, character, and business integrity.

Intro.2.2.4 Grounds for Decertification

Revocation of a certification, or decertification, results from a certificant failing to reasonably adhere to the policies and procedures of Software Certifications as defined by the Software Certifications Board. The Board may revoke certification for the following reasons:

- Falsifying information on the initial application and/or a CPE reporting form,
- Failure to abide by and support the Software Certifications Code of Ethics,
- Failure to submit the required continuing education credits toward recertification as required, or
- Failure to submit the required recertification fees as required.

Upon revocation, the certificant is requested to return their current certification credentials. A certificant may appeal a revocation at any time by communicating, in writing, directly with the Board.

Intro.2.3 Submitting the Initial Application

A completed Certification Candidacy Application must be submitted for entrance to Software Certifications as a candidate for any particular certification. Software Certifications strongly recommends that you submit your application only if you are prepared to sit and pass the CSTE examination. Submit the application only if you have:

- Satisfied all of the prerequisites for candidacy as stated on page 7.
- Subscribed to the Code of Ethics as described on page 9.
- Reviewed the CBOK and identified those areas that require additional studying.

The entire CBOK is provided in Skill Category 1 through Skill Category 10. A comprehensive list of related references is listed in Appendix B.

- Current experience in the field covered by the certification designation.
- Significant experience and breadth to have mastered the basics of the entire CBOK.
- Prepared to take the required examination and therefore ready to schedule and take the examination.

It should not be submitted by individuals who:

- Have not met all of the requirements stated above.
- Are not yet working in the field but who have an interest in obtaining employment in the field.
- Are working in limited areas of the field but would like to expand their work roles to include broader responsibilities.
- Are working in IT but have only marginal involvement or duties related to the certification.

- Are interested in determining if this certification program will be of interest to them.

Candidates for certification who rely on only limited experience, or upon too few or specific study materials, typically do not successfully obtain certification. Many drop out without ever taking the examination. Fees in this program are nonrefundable.

Do not apply unless you feel confident that your work activities and past experience have prepared you for the examination process.

Applicants already holding a certification from Software Certifications must still submit a new application when deciding to pursue an additional certification. For example, an applicant already holding a CSQA or CSPM certification must still complete the application process if pursuing the CSTE certification.

All supporting application forms and required fees must be filed with Software Certifications *at least 60 calendar days* prior to any examination date selected. The candidate must sign the application form agreeing to support and abide by the Software Certifications Code of Ethics. Applications will not be processed if they are incomplete, incorrectly completed, or fees have not been paid. See www.softwarecertifications.org for application fee information. The candidate has sole responsibility to ensure that materials are submitted in a timely and orderly manner.

When sending an application, please allow two weeks for processing. There is no need to contact the administrative office during this period to check on the status of the application. In fact, to protect the integrity of the examination and certification processes, all correspondence related to certification policies and procedures must be in writing, using e-mail, fax, or first-class postal service. Information and status obtained through telephone conversations with the administering body shall be considered unofficial and off-the-record.

Intro.2.3.1 Correcting Application Errors

The accuracy and correctness of applications, documentation, or payments are the responsibility of the applicant. Incomplete or erroneous paperwork is returned to the applicant for correction and resubmission. Common defects requiring paperwork to be returned to the applicant include:

- Required information is missing
- Incorrect form was used
- Payment is missing or invalid
- Unable to read required portions of application
- Required signature is not present
- Application received too late to be processed for selected examination

Once corrected, materials can be resubmitted. This correction cycle does not waive the requirement that all processing be completed at Software Certifications at least 60 days before any scheduled examination. Applicants are strongly advised to not delay submission of materials until close to that deadline.

Intro.2.3.2 Submitting Application Changes

It is critical that candidates submit changes to their candidacy application and keep their program records up-to-date. Many candidates change their residence or job situations during their certification candidacy. Others change their name as a result of marriage or divorce. If any such changes occur, it is the candidate's responsibility to notify the certification administrator using the Change of Records Form.

Intro.2.4 Application-Examination Eligibility Requirements

The candidate must take the initial exam within 12 months after acceptance. After the 12-month period, the candidate must resubmit the application, supporting documents, and any additional fees that may have been incurred. *A second or third sitting, if required, must be completed within 24 months of acceptance of the original application*. After the 24-month period, the candidate must reapply for candidacy to begin the process again.

The candidate may withdraw from the CSTE program at any time by submitting a Candidate Withdrawal Form to the certification administrator.

Candidates for certification must pass a four-part written examination in order to obtain certification. The examination tests the candidate's knowledge and practice of the competency areas defined in the CBOK. *Candidates who do not successfully pass the examination may re-sit for the examination up to two times* by submitting an Examination Retake Application (see Filing a Retake Application Filing a Retake Application) and paying all required fees. Subsequent additional examination efforts require reinitiating the entire application process.

The Software Certifications Board requires unsuccessful candidates to wait six months or more between examination sittings. Candidates who rapidly resit for examination parts are rarely successful. Adequate study and learning time needs to be spent in order to resit for missed examination parts successfully.

Technical knowledge becomes obsolete quickly; therefore the board has established these eligibility guidelines. The goal is to test on a consistent and comparable knowledge base worldwide. The eligibility requirements have been developed to encourage candidates to prepare and pass all portions of the examination in the shortest time possible.

Intro.2.4.1 Filing a Retake Application

A written Examination Retake Application must be submitted for each desired retake. As with the initial application, the application to reschedule and associated fees must be filed with

Software Certifications *at least 60 calendar days* before any examination date is selected. See www.softwarecertifications.org for application fee information.

Intro.3 Arranging to Sit and Take the Examination

When you have met all of the prerequisites as described above, you are ready to arrange to sit (or schedule) and take the CSTE examination. See "Preparing for the CSTE Examination" for information on what you need to do once you have scheduled the examination. This section also includes a sample examination with answers.

To schedule the CSTE examination, every candidate must:

- Satisfy all of the qualifications as described in "Meeting the CSTE Qualifications" starting on page Intro-7. Be certain that you are prepared and have studied the CBOK, the vocabulary in Appendix A, and the references in Appendix B.
- Schedule to take the examination. If you've studied enough that you feel you can commit to a specific examination date, visit www.softwarecertifications.org for dates or call Software Certifications. CSTE examinations are administered in various cities in the United States and all over the world. Submit a complete Examination Selection Form.
- Follow up on your examination schedule. After scheduling your examination you should receive a Confirmation Letter for the specific examination you indicated on your Examination Selection Form. See "Receiving the Confirmation Letter" on page Intro-15. Check www.softwarecertifications.org for your specific scheduled examination during the days leading up to the examination sitting for any changes to the schedule.
- Be sure to arrive at the examination early. See "Arriving at the Examination Site" on page 16 for a few tips, and what happens if you do not show up as scheduled.

Intro.3.1 Scheduling to Take the Examination

When you believe you are close to being prepared to take the examination, schedule to take the examination. To select an examination date and location that meets your needs submit an Examination Selection Form. Public certification examinations are scheduled periodically throughout the United States. A complete up-to-date schedule is on the Software Certifications Web site: *see Current Examination Schedule at www.softwarecertifications.org.*

Examination seating is limited, and seats are assigned on a first-come, first-served basis. *An Examination Selection Form must be submitted at least 60 days before the selected examination date in order to reserve a seat in the selected examination.* The earlier you apply the better chances of reserving a seat. The examination schedule can change on a weekly basis, so check www.softwarecertifications.org for any changes.

Examinations are held primarily by QAI Federation chapters, at major QAI conference programs, and by local QAI affiliates around the world. It is recommended that you contact Software Certification for site requirements, fees, and other details.

The certification examinations are typically available in Australia, Canada, Hong Kong, India, New Zealand, Saudi Arabia, Singapore, South Africa, United Arab Emirates, and the United States. As the worldwide acceptance of Software Certifications designations continues to grow, more locations will be hosting the exam. Please contact www.softwarecertification.org to inquire about examination locations.

Intro.3.1.1 Rescheduling the Examination Sitting

From time to time, candidates need to reschedule their intended examination date. This is known as a deferral, and is accomplished using the *Examination Deferral Form that must be submitted to the certification administrator at least 30 days before the originally scheduled examination*. If done in this manner, the *Examination Selection Form can be used to schedule the new examination as long as it is received at least 60 days before the new requested date*.

Deferrals received within 30 days of an examination date cannot be processed because examination materials have already been sent to the field. These candidates are considered "no shows" on the day of the examination and must use the Examination Retake Application in order to schedule a new examination date. As with the initial application, the *Examination Retake Application and associated fees must be filed with Software Certifications at least 60 days before any examination date is selected.*

Intro.3.2 Receiving the Confirmation Letter

Each candidate should receive an Confirmation Letter. You should bring this letter to the examination site along with photo identification to gain entry. When the letter is received, verify the examination information to assure that you have been scheduled for the examination selected, and that your contact information is all correct. If not received three weeks before a scheduled sitting, check the Current Examination Schedule for possible changes, or contact Software Certifications via e-mail for confirmation or correction.

Intro.3.3 Checking Examination Arrangements

Candidates are strongly encouraged to check www.softwarecertifications.org for your specific scheduled examination during the days leading up to the examination sitting. While Software Certifications makes every possible effort to provide examinations as scheduled, last minute changes have been sometimes unavoidable in the past. Previous disruptions have included inclement weather and building closures. The Current Examination Schedule is kept as up-to-date as possible when such situations occur.

Intro.3.4 Arriving at the Examination Site

Candidates should arrive at the examination location at least 30 minutes before the scheduled start time of the examination. Candidates must have their confirmation letter and photo identification with them in order to register and gain admission to the examination.

Intro.3.4.1 No-shows

Candidates who fail to appear for a scheduled examination – initial or retake – automatically fail the examination and must submit the Examination Retake Application to apply for a new examination date. Candidates who have filed a deferral after the 30-day advance deadline are considered to be no-shows as well.

Intro.4 How to Maintain Competency and Improve Value

Maintaining your personal competency is too important to leave to the soul discretion of your employer. In today's business environment you can expect to work for several different organizations, and to move to different jobs within your own organization. In order to be adequately prepared for these changes you must maintain your personal competency in your field of expertise.

Intro.4.1 Continuing Professional Education

Most professions recognize that a minimum of 40 hours of continuing professional education is required to maintain competency of your skills. There are many ways to get this training, including attending professional seminars and conferences, on-the-job training, attending professional meetings, taking e-learning courses, and attending professional association meetings.

You should develop an annual plan to improve your personal competencies. Getting 40 hours of continuing professional education will enable you to recertify your CSTE designation, but it will not necessarily improve your competencies. For example, you may get 24 hours CPE credit for attending a 3-day seminar, but if you're already competent in the seminar topic, it will not add to your personal capabilities.

The Common Body of Knowledge (CBOK) for the CSTE should be your guide for improving your personal competencies. A self-assessment of your competencies in the CBOK is provided in "CSTE 2006 Skill Assessment Worksheet." This assessment is designed to help you identify areas in which additional training would be beneficial to both you and your employer. After taking this competency assessment, you can use the results to create a

personal plan of action for you to ensure that you maintain the necessary professional competency to prepare you for change and/or promotion.

Intro.4.2 Advanced CSTE Designations

You can use your continuing professional education plan to improve and demonstrate your value to your employer. You can obtain your professional education credits while applying for an advanced certification. Your employer may have difficulty assessing improved competencies attributable to the continuing professional education you are acquiring. However, if you can use that continuing education effort to obtain an advanced certification, you can demonstrate to your employer your increased value to the organization by acquiring an advanced certification

There are two levels of advanced certifications you will be eligible for once you obtain your CSTE designation:

- Advanced Software Tester (ASTE)

  This advanced designation is designed to demonstrate your knowledge of how to do the testing tasks you may be assigned. The CSTE designation is focused much more on "what" you must know in order to practice testing. The ASTE designation is designed for those who can demonstrate they know "how" to perform testing tasks.

- Master Software Tester (MSTE)

  This is the highest designation attainable in the IT testing field. It is reserved for those who can demonstrate testing qualities and professional responsibilities.

The drivers for improving performance in IT are the quality assurance and quality control (testing) professionals. Dr. W. Edward Deming recognized this "do-check" partnership of quality professionals in his "14 points" as the primary means for implementing the change needed to mature. Quality control identifies the impediments to quality and quality assurance facilitates the fix. Listed below is the certification level, emphasis of each certification, and how you can demonstrate that competency.

Intro.4.2.1 What is the Certification Competency Emphasis?

- CSTE

  Demonstrate competency in knowing what to do.

  Study for, and pass, a four-part examination developed by peers to evaluate the candidate's knowledge of the principles and concepts incorporated into the CBOK, plus the ability to relate those principles and concepts to the challenges faced by IT organizations.

- ASTE

  Demonstrate competency in knowing how to do it.

  Candidates must demonstrate their ability to develop real solutions to challenges in their IT organizations, by proposing a solution to a real-world problem. This must be done for five CBOK categories, where each proposed solution must be accepted by the

Certification Board. Each accepted solution will be awarded a certificate of competency for that CBOK category.

- MSTE

Demonstrate competency in knowing how to break through testing and productivity barriers.

Candidates must demonstrate the ability to innovate beyond current practice in solving IT challenges, as well as, demonstrate public service in the IT testing profession. (Note: this certification available starting in 2006.)

Figure Intro-1 illustrates how you can improve your personal competencies.

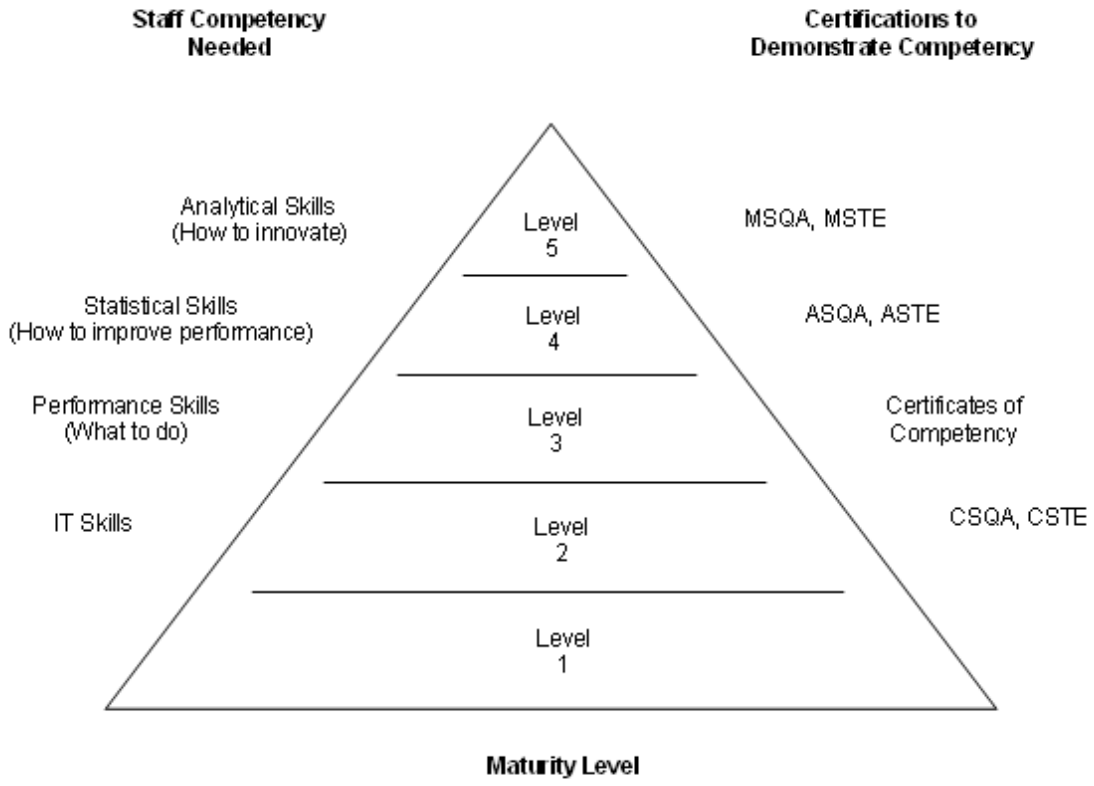


Figure Intro-1 Maturing Your Professional Competencies

For more information on the type of training that is applicable toward your continuing professional education requirements, and information on the advanced testing certifications and how to apply for them, visit www.softwarecertifications.org.

Preparing for the CSTE Examination

The CSTE examination is designed to evaluate your knowledge of the principles and practices of software testing. The principles primarily will involve vocabulary. This is to ensure that you understand what quality in an IT function is attempting to accomplish. The second half of the examination is on the application of those principles. This is to ensure that you can recognize good software testing practices when they occur.

Preparing for any standardized examination should be considered a serious undertaking. Begin preparing and studying well in advance. Remember that the minimum requirement for submitting your application is 60 calendar days prior to the exam date. When you know you will be applying for the examination, submit your application and fees and begin studying. Avoid "cramming," as it is rarely beneficial in the long term. See the "Introduction" for detailed information on submitting your application.

Assess Your CSTE 2006 CBOK Competency	**Intro-19**
Understand the Key Principles Incorporated Into the Examination	**Intro-22**
Review the List of References	**Intro-23**
Initiate a Self-Study Program	**Intro-24**
Take the Sample Examination	**Intro-24**

Intro.5 Assess Your CSTE 2006 CBOK Competency

The Common Body of Knowledge (CBOK) for the CSTE is in effect a job description for a world-class IT software tester. The CSTE Certification Board has defined the skills within the CBOK as those skills that would enable an IT software tester to perform the tasks needed to meet today's IT testing challenges.

Many human resource organizations use the CSTE CBOK as the basis for writing job descriptions for IT software testers. To properly prepare yourself to be proficient in the practice of IT testing, you should develop a personal plan of action that would enable you to assess your competency in the 2006 CSTE CBOK. It is recognized that many software testers do not need to be competent in all of the skill categories to fulfill their current job responsibilities.

The current CSTE CBOK includes ten skill categories that are fully described in this guide:

Skill Category 1- Software Testing Principles and Concepts

Skill Category 2- Building the Test Environment

Skill Category 3- Managing the Test Project

Skill Category 4- Test Planning

Skill Category 5- Executing the Test Plan

Skill Category 6- Test Reporting Process

Skill Category 7- User Acceptance Testing

Skill Category 8- Testing Software Developed by Contractors

Skill Category 9- Testing Software Controls and the Adequacy of Security Procedures

Skill Category 10- Testing New Technologies

Skill Categories 1-10 are common to all testing-related assignments and therefore, the certification examination focuses on all categories equally.

Intro.5.1 Complete the CSTE Skill Assessment Worksheet

To assess your competency of the CSTE CBOK, complete the worksheet, "CSTE 2006 Skill Assessment Worksheet" starting on page +21) 29 Follow these guidelines on how to use the worksheet to rate your competency and identify those areas that you need to better understand to successfully pass the CSTE examination:

1. Assess your competency of each skill listed on the worksheet. Carefully read each skill within the skill category. Based on your reading of the skill, assess your competency in one of the following three categories and place a checkmark ("✓") in the appropriate column on the CSTE 2006 CBOK Competency Rating Table:

   Not Competent – "None"

   Either you do not understand this skill, or if you do understand it you do not know "what" is required to perform this skill. For example, you may know that an IT test plan is needed, but you do not know what is included in an IT test plan.

   Some Competency – "Some"

   This assessment means that you know "what" is needed to accomplish a specific skill. For example, you may know what is to be included within an IT test plan, but you have never actually prepared an IT test plan. In other words, you have book knowledge, but not how-to knowledge.

Fully Competent – "Full"

This assessment means that you not only know what is required to perform a specific skill, but you have actually used that skill in performing day-to-day work tasks. For example, you have written an IT test plan.

Note that Skill Category 1 focuses on the vocabulary of IT software testing and the basic concepts on which the software testing profession is built. In assessing this category for a testing term such as reliability a "not competent" response means you could not define the term; a "some competency" response means you could define the term; and a "fully competent" response means that you use the term in the performance of your day-to-day work.

2. Study those skills you rated "None." After you complete the assessment worksheet, you will have designated some of the skills included in the CBOK as: None, Some, and Full. The objective in preparing for the CSTE examination should be to have "some competency" in all of the skills within the CBOK. You need not be fully competent in any skill to qualify you to pass the CSTE examination.

   Note that the CSTE designation focuses on individuals knowing "what to do" in order to effectively perform IT software testing. To provide maximum value to your employer, and to enable you to obtain either an Advanced Software Tester (ASTE) or Master Software Tester (MSTE) designation you need to be "fully competent" in most of the CBOK skills areas.

3. Reassess those skills you studied after a rating of "None." If you now believe your rating changes to "Some," then change your checkmark for the related skill on that category assessment table. Continue reassessing as you study.

Proceed only when you believe you are ready to submit your application for the CSTE certification examination.

Intro.5.2 Calculate Your CSTE CBOK Competency Rating

Follow these steps to calculate your competency rating for the CSTE 2006 CBOK. This rating will help you determine if you are ready to submit your application for the CSTE examination or if, and in what areas, you need further study in order to pass the examination. Use the CBOK Skill Category Competency Rating Table on page 40 to perform each step below.

1. Total the number of skills you have checked in each of the three columns for each skill category. Write your numbers in the space provided for each skill category on the worksheet. These are your competency rating totals for that skill category.

2. Transfer the three competency rating totals for each skill category to the corresponding column ("Full," "Some," and "None") in the CSTE Skill Category Competency Ratings table provided.

3. Tally each column in the table to determine each Ratings Total.
4. Multiply each column by the indicated number to determine the Column Total.
5. Add the Column Totals together to determine the Sum of the Rows Total.
6. Divide the Sum of the Rows Total by 160 (the number of total skills in the CSTE 2006 CBOK) to determine your CSTE CBOK Competency Rating. This number will be between 1 and 3.

Now you are able to determine if you are ready to submit your application and take the certification examination or if you need further study. Use your CSTE 2006 CBOK Competency Rating from step 6 above and the following key to interpret your competency rating:

- The closer your score is to "3," the more competent you are in software testing.
- If your score is a "3," you are a world-class software tester and ready to submit your application.
- If your score is between "2" and "3", you are a competent tester and ready to submit your application.
- See the "Introduction" for information on submitting your application for the CSTE 2006 certification examination.
- If your score is between "1" and "2", you do not have the basic skills necessary to perform software testing. Study those skills that you rated "None" and then reassess your skills.
- If your score is a "1", you are not competent in the CBOK. Study those skills that you rated "None" and then reassess your skills.

Using this product does not constitute, nor imply, the successful passing of the CSTE certification examination.

Intro.6 Understand the Key Principles Incorporated Into the Examination

This step is to provide you some insight into what will be emphasized on the examination. This should not be used in place of the CBOK. It is intended to emphasize some of the key concepts included within the CBOK.

In studying these key principles, two guidelines should be followed:

- Learn the vocabulary.

A major part of the CSTE examination and a major part of being an effective software tester is to understand and use the testing vocabulary. If you do not know the testing vocabulary, study Appendix A, "Vocabulary," before beginning any other CSTE examination preparatory activity. Note that understanding the vocabulary is essential to pass the examination.

- Learn how to apply the testing principles to everyday practice.

As you study the testing principles, think carefully how you would apply those principles to your day-to-day work challenges.

Intro.7 Review the List of References

Use the following lists of references to help you prepare for the CSTE examination:

- Appendix B of this preparation guide lists numerous books recommended in the software testing field.
- Software Certifications Web site – www.softwarecertifications.org (click on Index and then click on Body of Knowledge, CSTE) lists references compiled by the Certification Board and used in preparing the examination.

It is each candidate's responsibility to stay current in the field and to be aware of published works and materials available for professional study and development. Software Certifications recommends that candidates for certification continually research and stay aware of current literature and trends in the field. The lists referenced above are suggestions; they are not intended to be all-inclusive.

Use these lists of references in the following ways:

- Search your library for availability.

If you have these books in your reference library, company library, or ready access, set them aside for exam preparation.

- Use your assessment results (e.g., skills marked "Not Competent") from the previous step to determine which books would help you build your skills in those areas. Note that while studying, look for principles as opposed to learning detailed how-to skills.
- Review the list of references from the perspective of the types of materials that might be included on the examination. The references give you insight into the topics that will be included on the examination.

Intro.8 Initiate a Self-Study Program

This guide contains a variety of skill areas designed to be representative of the types of skills needed by software testers, and representative of the skill categories addressed in the CSTE examination. You may decide to start or join a self-study group in your area.

In developing a self-study program, you should:

- Assess your skills against the CSTE 2006 CBOK and complete the assessment worksheet.
- Study the key reference documents from the previous step. Use a representative sample of the reference books for study; if you do not have the specific reference book, use a similar book on the same topic.
- Attend formal courses, seminars, local testing-oriented chapter meetings, and testing conferences to gain a comprehension of the practice of testing. Be sure to visit www.qaiworldwide.org for up-to-date information on courses, seminars, and conferences. QAI offers a preparation course for the CSTE.

Self-study becomes more effective if you can work with one or more other candidates for the examination. If no other candidates are available to form a study group, locate a CSTE to become your mentor during your self-study period.

Intro.9 Take the Sample Examination

We have provided a sample CSTE examination for you to use in your preparations. See Appendix C “Preparing For the CSTE Examination” for the following useful information:

- CSTE Examination Overview, including how the test is structured and the number of questions, plus general information on how the test is administered at the test site.
- Guidelines to Answer Questions, including useful steps to answer all questions, tips on responses to essay questions, and what to do if you do not know the answer to a question.
- Sample CSTE Examination, including multiple-choice questions and essay questions. These give you examples of the types of questions on the examination. Also provided is an answer key to help you study and show you the types of essay responses expected.

This page intentionally left blank.

CSTE 2006 Skill Assessment Worksheet

Assess Your Skills against the CSTE 2006 CBOK

The 10 Skill Categories in the CSTE CBOK are common to all test-related assignments and therefore, the certification examination focuses equally on all of them.

The 2006 Common Body of Knowledge for the software tester certificate includes these ten skill categories:

Skill Category 1 – Software Testing Principles and Concepts

Skill Category 2 – Building the Test Environment

Skill Category 3 – Managing the Test Project

Skill Category 4 – Test Planning

Skill Category 5 – Executing the Test Plan

Skill Category 6 – Test Reporting Process

Skill Category 7 – User Acceptance Testing

Skill Category 8 – Testing Software Developed by Contractors

Skill Category 9 – Testing Internal Control

Skill Category 10 – Testing New Technologies

See Intro.5, Assess Your CSTE 2006 CBOK Competency on page Intro-19 for detailed instructions on how to use the worksheet and competency rating table.

Skill Category 1 – Software Testing Principles and Concepts

The "basics" of software testing are represented by the vocabulary of testing, testing approaches, methods and techniques as well as the materials used by testers in performing their test activities.

	Skill Category 1 – Software Testing Principles and Concepts	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
1.1	*Vocabulary* Quality Assurance versus Quality Control			
1.2	The Cost of Quality			
1.3	Software Quality Factors			
1.4	How Quality is Defined			
1.5	*Why Do We Test Software?* Developers are not Good Testers			
1.6	What is a Defect?			
1.7	What is Quality Software?			
1.8	Why Does a Development Process Produce Defects?			
1.9	Reducing the Frequency of Defects in Software Development			
1.10	*The Multiple Roles of the Software Tester* People Relationships			
1.11	Scope of Testing			
1.12	When Should Testing Occur?			
1.13	How the Test Plan Should be Developed			
1.14	Testing Constraints			
1.15	*Life Cycle Testing*			
1.16	*Test Matrices* Cascading Test Matrices			
1.17	*Independent Testing*			
1.18	*Tester's Workbench* What is a Process?			
1.19	*Levels of Testing* The "V" Concept of Testing			
1.20	*Testing Techniques* Structural versus Functional Technique Categories			
1.21	Verification versus Validation			
1.22	Status versus Dynamic Testing			
1.23	Examples of Specific Testing Techniques			
1.24	Combining Specific Testing Techniques			

Competency Rating Totals (total each "✓" in each column): _____ _____ _____

Skill Category 2 – Building the Test Environment

The test environment is comprised of all the conditions, circumstances, and influences surrounding and affecting the testing of software. The environment includes the organization's policies, procedures, culture, attitudes, rewards, test processes, test tools, methods for developing and improving test processes, management's support of software testing, as well as any test labs developed for the purpose of testing software and multiple operating environments. This category also includes assuring the test environment fairly represents the production environment to enable realistic testing to occur.

	Skill Category 2 – Building the Test Environment	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
2.25	*Management Support* Management Tone			
2.26	Integrity and Ethical Values			
2.27	Commitment to Competence			
2.28	Management's Philosophy and Operating Style			
2.29	Organizational Structure			
2.30	*Test Work Processes* The Importance of Work Processes			
2.31	Responsibility for Building Work Processes			
2.32	Developing Work Processes			
2.33	Tester's Workbench			
2.34	Analysis and Improvement of the Test Process			
2.35	*Test Tools* Tool Development and Acquisition			
2.36	Tool Usage			
2.37	Testers Competency			

Competency Rating Totals (total each "✓" in each column): _____ _____ _____

Skill Category 3 – Managing the Test Project

Software testing is a project with almost all the same attributes as a software development project. Software testing involves project planning, project staffing, scheduling and budgeting, communicating, assigning and monitoring work and ensuring that changes to the project plan are incorporated into the test plan.

	Skill Category 3 – Building the Test Environment	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
3.38	*Test Administration* Test Planning			
3.39	Customization of the Test Process			
3.40	Budgeting			
3.41	Scheduling			
3.42	Staffing			
3.43	*Test Supervision* Communication Skills			
3.44	Negotiation and Complaint Resolution Skills			
3.45	Judgment			
3.46	Providing Constructive Criticism			
3.47	Project Relationships			
3.48	Motivation, Mentoring and Recognition			
3.49	*Test Leadership* Chairing Meetings			
3.50	Team Building			
3.51	Quality Management Organizational Structure			
3.52	Code of Ethics			
3.53	*Managing Change* Software Configuration Management			
3.54	Change Management			

Competency Rating Totals (total each "✓" in each column): _____ _____ _____

Skill Category 4 – Test Planning

Testers need the skills to plan tests, including the selection of techniques and methods to be used to validate the product against its approved requirements and design. Test planning assesses the software application risks, and then develops a plan to determine if the software minimizes those risks. Testers must understand the development methods and environment to effectively plan for testing.

	Skill Category 4 – Building the Test Environment	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
4.55	*Risk Concepts and Vocabulary*			
4.56	*Risks Associated with Software Development*			
4.57	*Risks Associated with Software Testing* Premature Release Risk			
4.58	*Risk Analysis* Risk Analysis Process			
4.59	*Risk Management* Risk Reduction Methods			
4.60	Contingency Planning			
4.61	*Prerequisites to Test Planning* Test Objectives			
4.62	Acceptance Criteria			
4.63	Assumptions			
4.64	People Issues			
4.65	Constraints			
4.66	*Create the Test Plan* Understand the Characteristics of the Software Being Developed			
4.67	Build the Test Plan			
4.68	Write the Test Plan			

Competency Rating Totals (total each "✓" in each column): _____ _____ _____

Skill Category 5 – Executing the Test Plan

The test plan should be executed as designed. If the plan cannot be executed as designed it should be changed, or notations made as to what aspects of the plan were not performed. Testing according to the test plan should commence when the project commences and conclude when the software is no longer in operation. Portions of the test plan can be performed while the test plan is being written. Testers require many skills to carry out the test plan, like design test cases and test scripts, use test tools, execute tests, record test results, and manage defects.

	Skill Category 5 – Executing the Test Plan	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
5.69	*Test Case Design* Function Test Cases			
5.70	Structural Test Cases			
5.71	Erroneous Test Cases			
5.72	Stress Test Cases			
5.73	Test Scripts			
5.74	Use Cases			
5.75	*Building Test Cases* Process for Building Test Cases			
5.76	Example of Creating Test Cases for a Payroll Application			
5.77	*Test Coverage*			
5.78	*Performing Tests* Platforms			
5.79	Test Cycle Strategy			
5.80	Use of Tools in Testing			
5.81	Perform Tests			
5.82	When is Testing Complete?			
5.83	General Concerns			
5.84	*Recording Test Results* Problem Deviation			
5.85	Problem Effect			
5.86	Problem Cause			
5.87	Use of Test Results			
5.88	*Defect Management* Defect Naming			
5.89	The Defect Management Process			

Competency Rating Totals (total each "✓" in each column): _____ _____ _____

Skill Category 6 – Test Reporting Process

Testers need to demonstrate the ability to develop testing status reports. These reports should show the status of the testing based on the test plan. Reporting should document what tests have been performed and the status of those tests. The test reporting process is a process to collect data, analyze the data, supplement the data with metrics, graphs and charts and other pictorial representations which help the developers and users interpret that data. The lessons learned from the test effort should be used to improve the next iteration of the test process.

	Skill Category 6 – Test Reporting Process	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
6.90	*Prerequisites to Test Reporting* Define and Collect Test Status Data			
6.91	Define Test Metrics used in Reporting			
6.92	Define Effective Test Metrics			
6.93	*Test Tools used to Build Test Reports* Pareto Charts			
6.94	Pareto Voting			
6.95	Cause and Effect Diagrams			
6.96	Checksheets			
6.97	Histograms			
6.98	Run Charts			
6.99	Scatter Plot Diagrams			
6.100	Regression Analysis			
6.101	Multivariate Analysis			
6.102	Control Charts			
6.103	*Test Tools used to Enhance Test Reporting* Benchmarking			
6.104	Quality Function Deployment			
6.105	*Reporting Test Results* Current Status Test Reports			
6.106	Final Test Reports			
6.107	Guidelines for Report Writing			

Competency Rating Totals (total each "✓" in each column): _____ _____ _____

Skill Category 7 – User Acceptance Testing

The objective of software development is to develop the software that meets the true needs of the user, not just the system specifications. To accomplish this, testers should work with the users early in a project to clearly define the criteria that would make the software acceptable in meeting the users' needs. As much as possible, once the acceptance criterion has been established, they should integrate those criteria into all aspects of development. This same process can be used by software testers when users are unavailable for test; when diverse users use the same software; and for beta testing software. Although acceptance testing is a customer and user responsibility, testers normally help develop an acceptance test plan, include that plan in the system test plan to avoid test duplication; and, in many cases, perform or assist in performing the acceptance test.

	Skill Category 7 – User Acceptance Testing	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
7.108	*Acceptance Test Concepts* Difference between Acceptance Test and System Test			
7.109	*Roles and Responsibilities* User's Role			
7.110	Software Tester's Role			
7.111	*Acceptance Test Planning* Acceptance Criteria			
7.112	Acceptance Test Plan			
7.113	Use Case Test Data			
7.114	*Acceptance Test Execution* Execute the Acceptance Test Plan			
7.115	Acceptance Decision			

Competency Rating Totals (total each "✓" in each column): _____ _____ _____

Skill Category 8 – Testing Software Developed by Contractors

There are many challenges when testing software developed by a contractor, or an external organization. It is management's responsibility that acquired software meets the needs of their organization. Contractors will test the software they build, but that does not relieve management from their quality responsibilities. Management must put into place those test processes within their organization that provide the assurance that acquired software will perform as expected. Two test processes that are representative of best practices for testing acquired software are for COTS software and software developed under contract by an outside organization. Executing those defined test processes should be performed by software testers.

	Skill Category 8 – Testing Software Developed by Contractors	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
8.116	*Challenges in Testing Acquired Software* Purchased COTS Software			
8.117	Contracted Software			
8.118	*COTS Software Test Process* Assure Completeness of Needs Specification			
8.119	Define Critical Success Factor			
8.120	Determine Compatibility with Your Computer Environment			
8.121	Assure the Software can be Integrated into Your Business System Work Flow			
8.122	Demonstrate the Software in Operation			
8.123	Evaluate the People Fit			
8.124	Acceptance Test the COTS Software			
8.125	*Contracted Software Test Process* Assure the Process for Contracting Software is Adequate			
8.126	Review the Adequacy of the Contractor's Test Plan			
8.127	Assure Development is Effective and Efficient			
8.128	Perform Acceptance Testing on the Software			
8.129	Issue a Report on the Adequacy of the Software to Meet the Needs of the Organization			
8.130	Ensure Knowledge Transfer Occurs and Intellectual Property Rights are Protected			
8.131	Incorporate Copyrighted Material into the Contractor's Manuals			
8.132	Assure the Ongoing Operation and Maintenance of the Contracted Software			
8.133	Assure the Effectiveness of Contractual Relations			

Competency Rating Totals (total each "✓" in each column): _____ _____ _____

Skill Category 9 – Testing Internal Control

A key issue for software testing is testing internal control. Security is a component of internal control that warrants special attention of testers. Interest in internal control has been highlighted by publicized penetrations of security and the increased importance of information systems and the data contained by those systems. The passage of the Sarbanes-Oxley Act in particular, highlighted interest in internal control. The Sarbanes-Oxley Act, sometimes referred to as SOX, was passed in response to the numerous accounting scandals such as Enron and WorldCom. While much of the act relates to financial controls, there is a major section relating to internal controls. For Securities and Exchange Commission (SEC)-regulated corporations, both the CEO and the CFO must personally attest to the adequacy of their organization's system of internal control. Because misleading attestation statements is a criminal offense, top corporate executives take internal control as a very important topic. Many of those controls are incorporated into information systems, and thus the need for testing those controls.

	Skill Category 9 – Testing Internal Control	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
9.134	*Principles and Concepts of Internal Control* Internal control responsibilities			
9.135	Software Tester's Internal Controls Responsibilities			
9.136	Internal Auditor's Internal Control Responsibilities			
9.137	Risk versus Control			
9.138	Environmental versus Transaction Processing Controls			
9.139	Preventive, Detective and Corrective Controls			
9.140	*Internal Control Models* COSO Enterprise Risk Management (ERM) Model			
9.141	COSO Internal Control Framework Model			
9.142	CobiT Model			
9.143	*Testing Internal Controls* Perform Risk Assessment			
9.144	Test Transaction Processing Controls			
9.145	*Testing Security Controls* Task 1 – Where Security is Vulnerable to Penetration			
9.146	Task 2 – Building a Penetration Point Matrix			
9.147	Task 3 – Assess Security Awareness Training			
9.148	Task 4 – Understand the Attributes of an Effective Security Control			
9.149	Task 5 – Selecting Techniques to Test Security			

Competency Rating Totals (total each "✓" in each column): _____ _____ _____

Skill Category 10 – Testing New Technologies

Testers require skills in their organization's current technology, as well as a general understanding of the new information technology that might be acquired by their organization. The new technology skills are required because the test plan needs to be based on the types of technology used. Also technologies new to the organization and the testers pose technological risks which must be addressed in test planning and test execution. This section only addresses the newer IT technology, but any technology new to the testers or the organization must be addressed in the test plan.

	Skill Category 10 – Testing New Technologies	**Competency Rating**		
Skill #	**Skill Description**	**Full**	**Some**	**None**
10.150	*Risks Associated with New Technology*			
10.151	*Newer IT Technology that Impact Software Testing* Web-Based Applications			
10.152	Distributed Application Architecture			
10.153	Wireless Technologies			
10.154	New Application Business Models			
10.155	New Communication Methods			
10.156	Wireless Local Area Networks			
10.157	New Testing Tools			
10.158	*Testing the Effectiveness of Integrating New Technologies* Determine the Process Maturity Level of the New Technology			
10.159	Test the Controls over Implementing the New Technology			
10.160	Test the Adequacy of Staff Skills to Use the Technology			

Competency Rating Totals (total the "✓" in each column): _____ _____ _____

CSTE 2006 CBOK Competency Rating Table

CBOK Skill Category Competency Ratings			
	Full	**Some**	**None**
Skill Category 1			
Skill Category 2			
Skill Category 3			
Skill Category 4			
Skill Category 5			
Skill Category 6			
Skill Category 7			
Skill Category 8			
Skill Category 9			
Skill Category 10			
Ratings Total			
Factor for Multiplication	x 3	x 2	x 1
Columns Total			
Sum of the Rows Total			
Number of CSTE Skills			÷ 160
Your CSTE 2006 CBOK Competency Rating			

See Intro.5, Assess Your CSTE 2006 CBOK Competency on page Intro-19 for detailed instructions on how to use the worksheet and competency rating table.

Skill Category

1

Software Testing Principles and Concepts

The "basics" of software testing are represented by the vocabulary of testing, testing approaches, methods and techniques, as well as, the materials used by testers in performing their test activities.

Vocabulary	**1-1**
Why Do We Test Software?	**1-15**
Factors Affecting Software Testing	**1-29**
Life Cycle Testing	**1-40**
Test Matrices	**1-42**
Independent Testing	**1-45**
Tester's Workbench	**1-47**
Levels of Testing	**1-51**
Testing Techniques	**1-62**

1.1 Vocabulary

A unique characteristic of a profession is its vocabulary. The profession's vocabulary represents the knowledge of the profession and its ability to communicate with others about the professions knowledge. For example, in the medical profession one hundred years ago doctors referred to "evil spirits" as a diagnosis. Today the medical profession has added words such as cancer, AIDS, and stroke which communicate knowledge.

Appendix A in this study guide is a glossary of software testing terms. It is very important that testers know this vocabulary.

Three important vocabulary concepts are discussed in detail below:

- The difference between quality assurance and quality control
- The cost of quality
- Software quality factors

1.1.1 Quality Assurance Versus Quality Control

Testing is a Quality Control Activity.

There is often confusion in the IT industry regarding the difference between quality control and quality assurance. Many "quality assurance" groups, in fact, practice quality control. Quality methods can be segmented into two categories: preventive methods and detective methods. This distinction serves as the mechanism to distinguish quality assurance activities from quality control activities. This discussion explains the critical difference between control and assurance, and how to recognize a control practice from an assurance practice.

Quality has two working definitions:

- Producer's Viewpoint – The quality of the product meets the requirements.
- Customer's Viewpoint – The quality of the product is "fit for use" or meets the customer's needs.

There are many "products" produced from the software development process in addition to the software itself, including requirements, design documents, data models, GUI screens, programs, and so on. To ensure that these products meet both requirements and user needs, both quality assurance and quality control are necessary.

1.1.1.1 Quality Assurance

Quality assurance is a planned and systematic set of activities necessary to provide adequate confidence that products and services will conform to specified requirements and meet user needs. Quality assurance is a staff function, responsible for implementing the quality policy defined through the development and continuous improvement of software development processes.

Quality assurance is an activity that establishes and evaluates the processes that produce products. If there is no need for process, there is no role for quality assurance. For example, quality assurance activities in an IT environment would determine the need for, acquire, or help install:

- System development methodologies
- Estimation processes
- System maintenance processes
- Requirements definition processes
- Testing processes and standards

Once installed, quality assurance would measure these processes to identify weaknesses, and then correct those weaknesses to continually improve the process.

1.1.1.2 Quality Control

Quality control is the process by which product quality is compared with applicable standards, and the action taken when nonconformance is detected. Quality control is a line function, and the work is done within a process to ensure that the work product conforms to standards and requirements.

Quality control activities focus on identifying defects in the actual products produced. These activities begin at the start of the software development process with reviews of requirements, and continue until all application testing is complete.

It is possible to have quality control without quality assurance. For example, a test team may be in place to conduct system testing at the end of development, regardless of whether that system is produced using a software development methodology.

Both quality assurance and quality control are separate and distinct from the internal audit function. Internal Auditing is an independent appraisal activity within an organization for the review of operations, and is a service to management. It is a managerial control that functions by measuring and evaluating the effectiveness of other controls.

The following statements help differentiate quality control from quality assurance:

- Quality control relates to a specific product or service.
- Quality control verifies whether specific attribute(s) are in, or are not in, a specific product or service.
- Quality control identifies defects for the primary purpose of correcting defects.
- Quality control is the responsibility of the team/worker.
- Quality control is concerned with a specific product.

- Quality assurance helps establish processes.
- Quality assurance sets up measurement programs to evaluate processes.
- Quality assurance identifies weaknesses in processes and improves them.
- Quality assurance is a management responsibility, frequently performed by a staff function.
- Quality assurance is concerned with all of the products that will ever be produced by a process.
- Quality assurance is sometimes called quality control over quality control because it evaluates whether quality control is working.
- Quality assurance personnel should never perform quality control unless it is to validate quality control.

1.1.2 The Cost of Quality

When calculating the total costs associated with the development of a new application or system, three cost components must be considered. The Cost of Quality, as seen in Figure 1-1, is all the costs that occur beyond the cost of producing the product "right the first time." Cost of Quality is a term used to quantify the total cost of prevention and appraisal, and costs associated with the production of software.

The Cost of Quality includes the additional costs associated with assuring that the product delivered meets the quality goals established for the product. This cost component is called the Cost of Quality, and includes all costs associated with the prevention, identification, and correction of product defects.

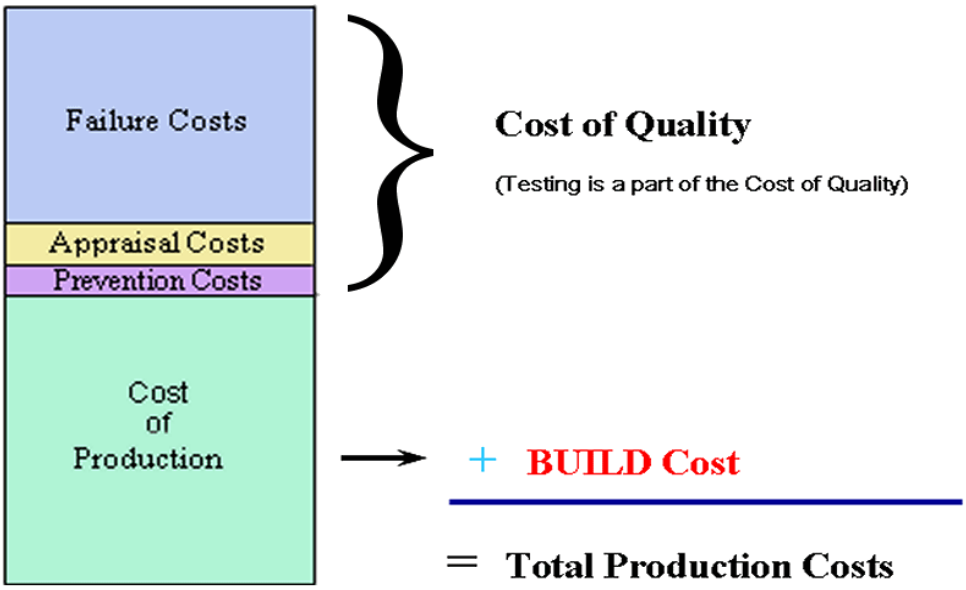


Figure 1-1 Cost of Quality

The three categories of costs associated with producing quality products are:

- Prevention Costs

  Money required to prevent errors and to do the job right the first time. These normally require up-front costs for benefits that will be derived months or even years later. This category includes money spent on establishing methods and procedures, training workers, acquiring tools, and planning for quality. Prevention money is all spent before the product is actually built.

- Appraisal Costs

  Money spent to review completed products against requirements. Appraisal includes the cost of inspections, testing, and reviews. This money is spent after the product is built but before it is shipped to the user or moved into production.

- Failure Costs

  All costs associated with defective products that have been delivered to the user or moved into production. Some failure costs involve repairing products to make them meet requirements. Others are costs generated by failures such as the cost of operating faulty products, damage incurred by using them, and the costs associated with operating a Help Desk.

The Cost of Quality will vary from one organization to the next. The majority of costs associated with the Cost of Quality are associated with the identification and correction of defects. To minimize production costs, the project team must focus on defect prevention. The goal is to optimize the production process to the extent that rework is eliminated and inspection is built into the production process. The IT quality assurance group must identify the costs within these three categories, quantify them, and then develop programs to minimize the totality of these three costs. Applying the concepts of continuous testing to the systems development process can reduce the cost of quality.

1.1.3 Software Quality Factors

In defining the scope of testing, the risk factors become the basis or objective of testing. The objectives for many tests are associated with testing software quality factors. The software quality factors are attributes of the software that, if they are wanted and not present, pose a risk to the success of the software, and thus constitute a business risk. For example, if the software is not easy to use, the resulting processing may be incorrect. The definition of the software quality factors and determining their priority enables the test process to be logically constructed.

When software quality factors are considered in the development of the test strategy, results from testing successfully meet your objectives.

The primary purpose of applying software quality factors in a software development program is to improve the quality of the software product. Rather than simply measuring, the concepts are based on achieving a positive influence on the product, to improve its development.

This section addresses the problem of identifying software quality factors that are in addition to the functional, performance, cost, and schedule requirements normally specified for software development. The fact that the goals established are related to the quality of the end product should, in itself, provide some positive influence.

Once the software quality factors have been determined by following the procedures described in the subsequent paragraphs, they must be transmitted to the development team. The software quality factors should be documented in the same form as the other system requirements and relayed to the development team. Additionally, a briefing emphasizing the intent of the inclusion of the software quality factors is recommended.

1.1.3.1 How to Identify Important Software Quality Factors

Figure 1-2 illustrates the Diagram of Software Quality Factors as describe by McCall in 1977. The basic tool used to identify the important software quality factors is the Software Quality Factor Survey form shown in Figure 1-3. The formal definitions of each of the eleven software quality factors are provided on that form.

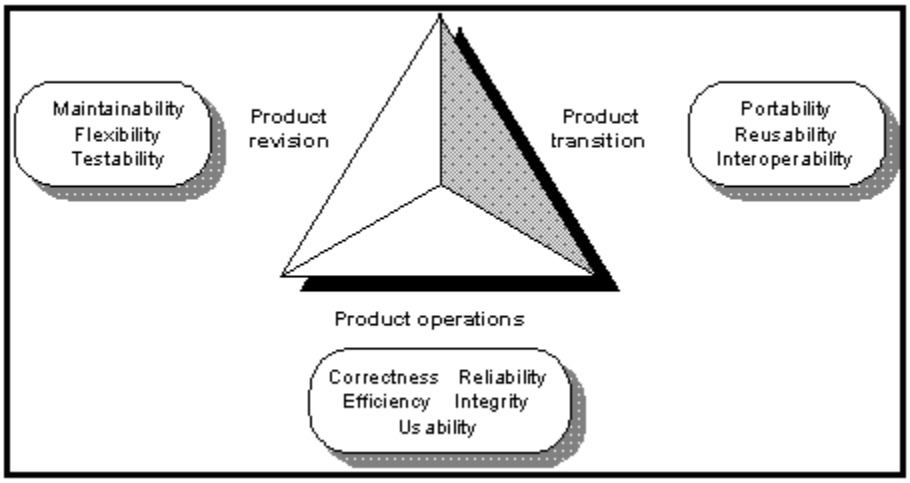


Figure 1-2 McCall's Diagram of Software Quality Factors

For your current system design goals, please rate each Software Quality Factor as: Very Important (VI), Important (I), Somewhat Important (SI), or Not Important (NI).		
Design Goal Rating	**Factors**	**Definition**
	Correctness	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
	Reliability	Extent to which a program can be expected to perform its intended function with required precision
	Efficiency	The amount of computing resources and code required by a program to perform a function.
	Integrity	Extent to which access to software or data by unauthorized persons can be controlled.
	Usability	Effort required learning, operating, preparing input, and interpreting output of a program
	Maintainability	Effort required locating and fixing an error in an operational program.
	Testability	Effort required testing a program to ensure that it performs its intended function.
	Flexibility	Effort required modifying an operational program.
	Portability	Effort required to transfer software from one configuration to another.
	Reusability	Extent to which a program can be used in other applications - related to the packaging and scope of the functions that programs perform.
	Interoperability	Effort required to couple one system with another.

Figure 1-3 Software Quality Factors Survey Form

It is recommended that you brief the decision makers using the tables and figures that follow in this section to solicit their responses to the survey. The decision makers may include the acquisition manager, the user or customer, the development manager, and the QA manager.

To complete the survey, follow these procedures:

1. Consider basic characteristics of the application.

   The software quality factors for each system are unique and influenced by system or application-dependent characteristics. There are basic characteristics that affect the software quality factors, and each software system must be evaluated for its basic characteristics. Figure 1-4 provides a list of some of these basic characteristics.

System Characteristic	Software Quality Factor
Human lives are affected	Reliability Correctness Testability
Long life cycle	Maintainability Flexibility Portability
Real-time application	Efficiency Reliability Correctness
On-board computer application	Efficiency Reliability Correctness
Processes classified information	Integrity
Interrelated systems	Interoperability

Figure 1-4 System Characteristics and Related Software Quality Factors

For example, if the system is being developed in an environment in which there is a high rate of technical breakthroughs in hardware design, portability should take on an added significance. If the expected life cycle of the system is long, maintainability becomes a cost-critical consideration. If the application is an experimental system where the software specifications will have a high rate of change, flexibility in the software product is highly desirable. If the functions of the system are expected to be required for a long time, while the system itself may change considerably, reusability is of prime importance in those modules that implement the major functions of the system.

With the advent of more computer networks and communication capabilities, more systems are being required to interface with other systems; hence, the concept of interoperability is extremely important. These and other system characteristics should be considered when identifying the important software quality factors.

2. Consider life cycle implications.

   The eleven software quality factors identified on the survey can be grouped according to three life cycle activities associated with a delivered software product, or post development. These three activities are product operation, product revision, and product transition. The relationship of the software quality factors to these activities is shown in Table 1-1. This table also illustrates where quality indications can be achieved through measurement and where the impact is felt if poor quality is realized. The size of this impact determines the cost savings that can be expected if a higher quality system is achieved through the application of the metrics.

A cost-to-implement versus life-cycle-cost reduction relationship exists for each software quality factor. The benefit versus cost-to-provide ratio for each factor is rated as high, medium, or low in the right-hand column of Table 1-1. This relationship and the life cycle implications of the software quality factors should be considered when selecting the important factors for a specific system.

Table 1-1: The Impact of Not Specifying or Measuring Software Quality Factors

	Development			Evaluation	Post-Development			
Life Cycle Phases / Software Quality Factors	Requirements Analysis	Design	Code & Debug	System Test	Operation	Revision	Transition	Expected Cost Saved vs. Cost to Provide
Correctness	Δ	Δ	Δ	X	X	X		High
Reliability	Δ	Δ	Δ	X	X	X		High
Efficiency	Δ	Δ	Δ		X			Low
Integrity	Δ	Δ	Δ		X			Low
Usability	Δ	Δ		X		X		Medium
Maintainability		Δ	Δ			X	X	High
Testability		Δ	Δ	X		X	X	High
Flexibility		Δ	Δ			X	X	Medium
Portability		Δ	Δ				X	Medium
Reusability		Δ	Δ				X	Medium
Interoperability	Δ	Δ		X			X	Low

LEGEND: Δ = Software quality factors should be measured
X = Impact of poor quality is realized

3. Perform trade-offs among the tentative list of software quality factors.

   As a result of the previous two steps, a tentative list of software quality factors should be produced. The next step is to consider the interrelationships among the factors selected. Figure 1-5 can be used as a guide for determining the relationships between the software quality factors. Some factors are synergistic while others conflict. The impact of conflicting factors is that the cost to implement will increase. This will lower the benefit-to-cost ratio described in the preceding paragraphs.

	Factors										
Factors	Correctness	Reliability	Efficiency	Integrity	Usability	Maintainability	Testability	Flexibility	Portability	Reusability	Interoperability
Correctness											
Reliability	↓										
Efficiency											
Integrity			↑								
Usability	↓	↓	↑	↓							
Maintainability	↓	↓	↑		↓						
Testability	↓	↓	↑		↓	↓					
Flexibility	↓	↓	↑	↑	↓	↓	↓				
Portability			↑			↓	↓				
Reusability		↑	↑	↑		↓	↓	↓	↓		
Interoperability			↑	↑					↓		

Legend
If a high degree of quality is present for the factor, what degree of quality is expected for the other:
↑ = High ↓ = Low
Blank = No relationship or application dependent

Figure 1-5 Relationship between Software Quality Factors

4. Identify most important software quality factors.

   The list of software quality factors considered to be important for the particular system compiled in the preceding three steps should be organized in order of importance. A single decision maker may choose the factors or the choice may be made by averaging several survey responses. The definitions of the factors chosen should be included with this list.

5. Provide explanations for choices.

   Document rationale for the decisions made during the first three steps.

1.1.3.2 Inventory Control System Example

To illustrate the application of these steps, consider an inventory control system. The inventory control system maintains inventory status and facilitates requisitioning, reordering, and issuing of supplies. The planned life of the system is ten years. Each step described previously will be performed with respect to the tactical inventory control system.

1. Consider Basic Characteristics of the Application.

   Utilizing Figure 1-4 and considering the unique characteristics of the tactical inventory control system, results in the following:

Table 1-2: System Characteristics and the Related Software Quality Factors

System Characteristic	Related Software Quality Factor
Critical Supplies	Correctness Reliability
Utilized by Supply Personnel	Usability
Long Life Cycle with Stable Hardware and Software Requirements	Maintainability
Interfaces with Inventory Systems at Other Sites	Interoperability

2. Consider life cycle implications.

   Of the five related software quality factors identified above, all provide high or medium life cycle cost benefits according to Table 1-1.

3. Perform trade-offs among factors.

   Using Figure 1-5, there are no conflicts that need to be considered.

4. Identify most important software quality factors.

   Using the survey form and the guidance provided in the preceding procedures, the following factors are identified in order of importance along with the definitions.

   - *Correctness* – Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
   - *Reliability* – Extent to which a program can be expected to perform its intended function with required precision.
   - *Usability* – Effort required learning, operating, preparing input, and interpreting output of a program.
   - *Maintainability* – Effort required locating and fixing an error in an operational program.
   - *Interoperability* – Effort required to couple one system to another.

5. Provide explanation of selected software quality factors.

   - *Correctness* – The system performs critical supply function.
   - *Reliability* – The system performs critical supply functions in remote environment.
   - *Usability* – The system will be used by personnel with minimum computer training.
   - *Maintainability* – The system life cycle is projected to be ten years and it will operate in the field where field personnel will maintain it.
   - *Interoperability* – The system will interface with other supply systems.

1.1.4 How Quality is Defined

The definition of "quality" is a factor in determining the scope of software testing. Although there are multiple quality philosophies documented, it is important to note that most contain the same core components:

- Quality is based upon customer satisfaction
- Your organization must define quality before it can be achieved
- Management must lead the organization through any improvement efforts

There are five perspectives of quality – each of which must be considered as important to the customer:

1. Transcendent – I know it when I see it
2. Product-Based – Possesses desired features
3. User-Based – Fitness for use
4. Development- and Manufacturing-Based – Conforms to requirements
5. Value-Based – At an acceptable cost

Peter R. Scholtes introduces the contrast between effectiveness (doing the right things) and efficiency (doing things right). Quality organizations must be both effective and efficient.

Patrick Townsend examines quality in fact and quality in perception as shown in Table 1-3. Quality in fact is usually the supplier's point of view, while quality in perception is the customer's. Any difference between the former and the latter can cause problems between the two.

Table 1-3: Townsend's Quality View

QUALITY IN FACT	QUALITY IN PERCEPTION
Doing the right thing.	Delivering the right product.
Doing it the right way.	Satisfying our customer's needs.
Doing it right the first time.	Meeting the customer's expectations.
Doing it on time.	Treating every customer with integrity, courtesy, and respect.

An organization's quality policy must define and view quality from their customer's perspectives. If there are conflicts, they must be resolved.

1.1.5 Definitions of Quality

Quality is frequently defined as meeting the customer's requirements the first time and every time. Quality is also defined as conformance to a set of customer requirements that, if met, result in a product that is fit for its intended use.

Quality is much more than the absence of defects, which allows us to meet customer expectations. Quality requires controlled process improvement, allowing loyalty in organizations. Quality can only be achieved by the continuous improvement of all systems and processes in the organization, not only the production of products and services but also the design, development, service, purchasing, administration, and, indeed, all aspects of the transaction with the customer. All must work together toward the same end.

Quality can only be seen through the eyes of the customers. An understanding of the customer's expectations (effectiveness) is the first step; then exceeding those expectations (efficiency) is required. Communications will be the key. Exceeding customer expectations assures meeting all the definitions of quality.

1.1.6 What is Quality Software?

There are two important definitions of quality software:

The producer's view of quality software means meeting requirements.

Customer's/User's of software view of quality software means fit for use.

These two definitions are not inconsistent. Meeting requirements is the producer's definition of quality; it means that the person building the software builds it in accordance with requirements. The fit for use definition is a user's definition of software quality; it means that the software developed by the producer meets the user's need regardless of the software requirements.

1.1.6.1 The Two Software Quality Gaps

In most IT groups, there are two gaps as illustrated in Figure 1-6, the different views of software quality between the customer and the producer.

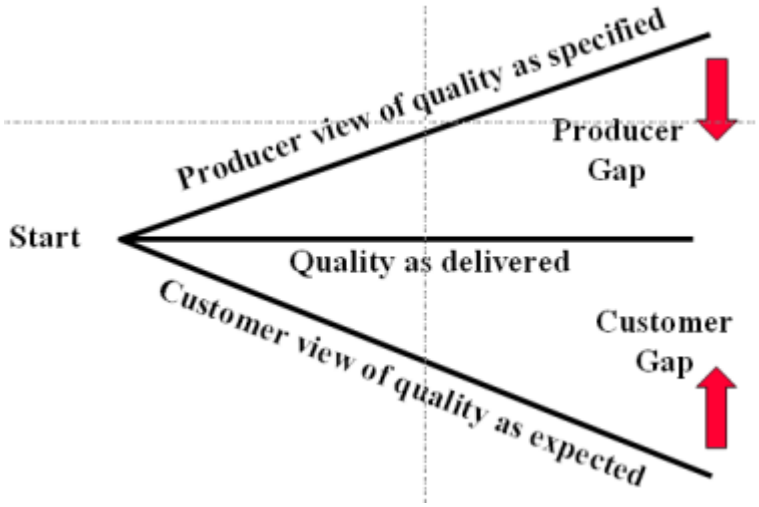


Figure 1-6 The Two Software Quality Gaps

The first gap is the producer gap. It is the gap between what is specified to be delivered, meaning the documented requirements and internal IT standards, and what is actually delivered. The second gap is between what the producer actually delivers compared to what the customer expects.

A potential role of software testing becomes helping to close the two gaps. The IT quality function must first improve the processes to the point where IT can produce the software according to requirements received and its own internal standards. The objective of the quality function closing the producer's gap is to enable an IT function to provide its user consistency in what it can produce. At QAI, we call this the "McDonald's effect." This means that when you go into any McDonald's in the world, a Big Mac should taste the same. It doesn't mean that you as a customer like the Big Mac or that it meets your needs, but rather, that McDonald's has now produced consistency in its delivered product.

To close the customer's gap, the IT quality function must understand the true needs of the user. This can be done by the following:

- Customer surveys
- JAD (joint application development) sessions – the producer and user come together and negotiate and agree upon requirements
- More user involvement while building information products

It is accomplished through changing the processes to close the user gap so that there is consistency, and producing software and services that the user needs. Software testing professionals can participate in closing these "quality" gaps.

1.1.6.2 What is Excellence?

The Random House College Dictionary defines excellence as "superiority; eminence." Excellence, then, is a measure or degree of quality. These definitions of quality and excellence

are important because it is a starting point for any management team contemplating the implementation of a quality policy. They must agree on a definition of quality and the degree of excellence they want to achieve.

The common thread that runs through today's quality improvement efforts is the focus on the customer and, more importantly, customer satisfaction. The customer is the most important person in any process. Customers may be either internal or external. The question of customer satisfaction (whether that customer is located in the next workstation, building, or country) is the essence of a quality product. Identifying customers' needs in the areas of what, when, why, and how are an essential part of process evaluation and may be accomplished only through communication.

The internal customer is the person or group that receives the results (outputs) of any individual's work. The outputs may include a product, a report, a directive, a communication, or a service. In fact, anything that is passed between people or groups. Customers include peers, subordinates, supervisors, and other units within the organization. Their expectations must also be known and exceeded to achieve quality.

External customers are those using the products or services provided by the organization. Organizations need to identify and understand their customers. The challenge is to understand and exceed their expectations.

An organization must focus on both internal and external customers and be dedicated to exceeding customer expectations.

1.2 What is Life Cycle Testing?

Testing is the process of evaluating a deliverable with the intent of finding errors.

Testing is NOT:

- A stage/phase of the project
- Just finding broken code
- A final exam
- “Debugging”

1.2.1 Why Do We Test Software?

The simple answer as to why we test software is that developers are unable to build defect-free software. If the development processes were perfect, meaning no defects were produced, testing would not be necessary.

Let's compare the manufacturing process of producing boxes of cereal to the process of making software. We find that, as is the case for most food manufacturing companies, testing each box of cereal produced is unnecessary. Making software is a significantly different process than making a box of cereal however. Cereal manufacturers may produce 50,000 identical boxes of cereal a day, while each software process is unique. This uniqueness introduces defects, and thus making testing software necessary.

As we introduce the skill categories of the CBOK for software testers, it is helpful to understand these six concepts, which explain why we test software:

- Developers are not good testers
- What is a defect?
- Why does a development process produce defects?
- Reducing the frequency of defects in software development
- An effective development process that minimizes defects
- How is quality defined?

1.2.2 Developers are not Good Testers

Testing by the individual who developed the work has not proven to be a substitute to building and following a detailed test plan. The disadvantages of a person checking their own work using their own documentation are as follows:

- Misunderstandings will not be detected, because the checker will assume that what the other individual heard from him was correct.
- Improper use of the development process may not be detected because the individual may not understand the process.
- The individual may be "blinded" into accepting erroneous system specifications and coding because he falls into the same trap during testing that led to the introduction of the defect in the first place.
- Information services people are optimistic in their ability to do defect-free work and thus sometimes underestimate the need for extensive testing.

Without a formal division between development and test, an individual may be tempted to improve the system structure and documentation, rather than allocate that time and effort to the test.

1.2.3 What is a Defect?

A defect is an undesirable state. There are two types of defects: process and procedure. For example, if a Test Plan Standard is not followed, it would be a process defect. However, if the Test Plan did not contain a Statement of Usability as specified in the Requirements documentation it would be a product defect.

The term quality is used to define a desirable state. A defect is defined as the lack of that desirable state. In order to fully understand what a defect is we must understand quality.

1.2.4 Software Process Defects

Ideally, the software development process should produce the same results each time the process is executed. For example, if we follow a process that produced one function-point-of-logic in 100 person hours, we would expect that the next time we followed that process, we would again produce one function-point-of-logic in 100 hours. However, if we follow the process the second time and it took 110 hours to produce one function-point-of-logic, we would state that there is "variability" in the software development process. Variability is the "enemy" of quality – the concepts behind maturing a software development process is to reduce variability.

The concept of measuring and reducing variability is commonly called statistical process control (SPC). To understand SPC we need to first understand the following:

- What constitutes an in control process
- What constitutes an out of control process
- What are some of the steps necessary to reduce variability within a process

Testers need to understand process variability, because the more variance in the process the greater the need for software testing. Following is a brief tutorial on processes and process variability.

1.2.4.1 What Does It Mean For a Process To Be In or Out of Control?

The amount of variation in a process is quantified with summary statistics; typically, the standard deviation is used. A process is defined as stable if its parameters (i.e., mean and standard deviation) remain constant over time; it is then said to be in a state of statistical control. Figure 1-7 illustrates a stable process. Such a process is predictable, i.e., we can predict, within known limits and with a stated degree of belief, future process values. Accepted practice uses a prediction interval three standard deviation distances in width around the population mean ($\mu \pm 3$) in establishing the control limits.

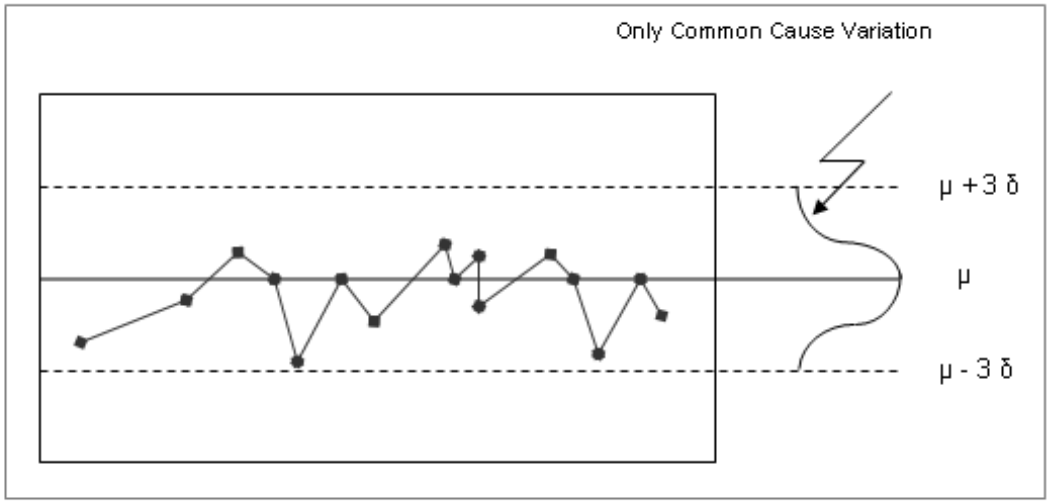


Figure 1-7 An In-Control (Stable) Process

Continuous process improvement through the use of quantitative methods and employee involvement sets quality management apart from other attempts to improve productivity. Continuous process improvement is accomplished by activating teams and providing them with quantitative methods such as SPC techniques and supporting them as they apply these tools. We will further discuss the concept of variation, common and special causes of variation, and QAI's Continuous Improvement Strategy.

The natural change occurring in organizational life moves systems and processes towards increasing variation. Statistical methods help us collect and present data in ways that facilitate the evaluation of current theories and the formation of new theories. These tools are the only methods available for quantifying variation. Since the key to quality is process consistency, variation (the lack of consistency) must be understood before any process can be improved. Statistical methods are the only way to objectively measure variability. There is no other way!

Variation is present in all processes. Table 1-4 lists some sources of variation for administrative processes.

Table 1-4: Typical Sources of Variation

Measurement Components Counting Sampling	**Material Components** Forms Suppliers
Machine Components Office equipment Computers Software	**Method Components** Procedures Policies Accounting practices
People Components Training Experience Attitude Aptitude	**Environment Components** Temperature Humidity Noise Level Lighting

The cumulative effect of sources of variation in a production process is shown in the table. One of the challenges in implementing quality management is to get those working in the process thinking in terms of sources of variation. How much of the observed variation can be attributed to measurements, material, machines, methods, people and the environment?

Consistency in all the processes from conception through delivery of a product or service is the cornerstone of quality. Paradoxically, the route to quality is not just the application of SPC and the resulting control charts. Managers must change the way they manage. They must use statistical methods in making improvements to management processes as well as all other processes in the organization.

Special causes of variation are not typically present in the process. They occur because of special or unique circumstances. If special causes of variation exist, the process is unstable or unpredictable. Special causes must be eliminated to bring a process into a state of statistical control. A state of statistical control is established when all special causes of variation have been eliminated.

Brian Joiner[1] has summarized special causes of variation as follows:

- Process inputs and conditions that sporadically contribute to the variability of process outputs.
- Special causes contribute to output variability because they themselves vary.
- Each special cause may contribute a “small” or “large” amount to the total variation in process outputs.
- The variability due to one or more special causes can be identified by the use of control charts.
- Because special causes are “sporadic contributors,” due to some specific circumstances, the “process” or “system” variability is defined without them.

Joiner then presents this strategy for eliminating special causes of variation:

1. Joiner, Brian, “Stable and Unstable Processes, Appropriate and Inappropriate Managerial Action.” From an address given at a Deming User’s Group Conference in Cincinnati, OH.

- Work to get very timely data so that special causes are signaled quickly – use early warning indicators throughout your operation.
- Immediately search for the cause when the control chart gives a signal that a special cause has occurred. Find out what was different on that occasion from other occasions.
- Do not make fundamental changes in that process.
- Instead, seek ways to change some higher-level systems to prevent that special cause from recurring. Or, if results are good, retain that lesson.

Common causes of variation are typically due to a large number of small random sources of variation. The sum of these sources of variation determines the magnitude of the process's inherent variation due to common causes; the process's control limits and current process capability can then be determined. Figure 1-8 illustrates an out of control process.

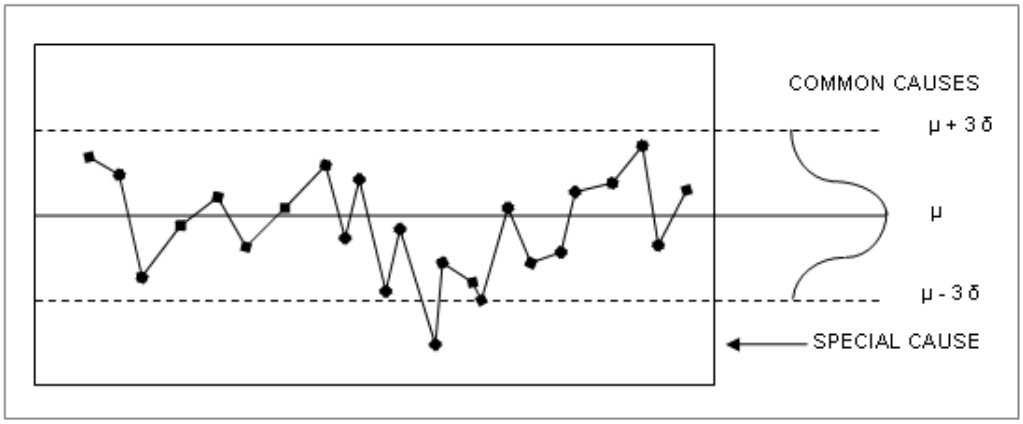


Figure 1-8 An Out-of-Control (Unstable) Process

Joiner also provides thoughts on common causes of variation:

- Process inputs and conditions that regularly contribute to the variability of process outputs.
- Common causes contribute to output variability because they themselves vary.
- Each common cause typically contributes a small portion to the total variation in process outputs.
- The aggregate variability due to common causes has a "nonsystematic," random-looking appearance.
- Because common causes are "regular contributors," the "process" or "system" variability is defined in terms of them.

Joiner also outlined a strategy for reducing common causes of variation:

- Talk to lots of people including local employees, other managers, and staff from various functions.
- Improve measurement processes if measuring contributes too much to the observed variation.

- Identify and rank categories of problems by Pareto analysis (a ranking from high to low of any occurrences by frequency).
- Stratify and desegregate your observations to compare performance of sub-processes.
- Investigate cause-and-effect relations. Run experiments (one factor and multifactor).

Those working in the process (employees) have the lead responsibility for the reduction of special causes of variation. Those working on the process (management) are responsible for leading the effort to reduce common cause variation. These higher-level improvements to the process usually require process or system changes. It is now widely recognized that as much as 94% of problems in any organization are system problems and the responsibility of management to solve. The concept of statistical control allows us to determine which problems are in the process (due to common causes of variation) and which are external to the process (due to special causes of variation).

Bringing a process into a state of statistical control is not really improving the process; it is just bringing it back to its typical operation. Reducing variation due to common causes is process improvement and the real essence of continuous process improvement.

As previously mentioned, variation due to special causes must be identified and removed to create a stable process. However, a stable process may not be an acceptable process. If its variation, due to common causes results in operation of the process beyond specifications, the process is called “incapable.” The process must be improved, i.e., variation due to common cause must be reduced or the process retargeted or both. Figure 1-9 illustrates the transition of a process from incapable to capable.

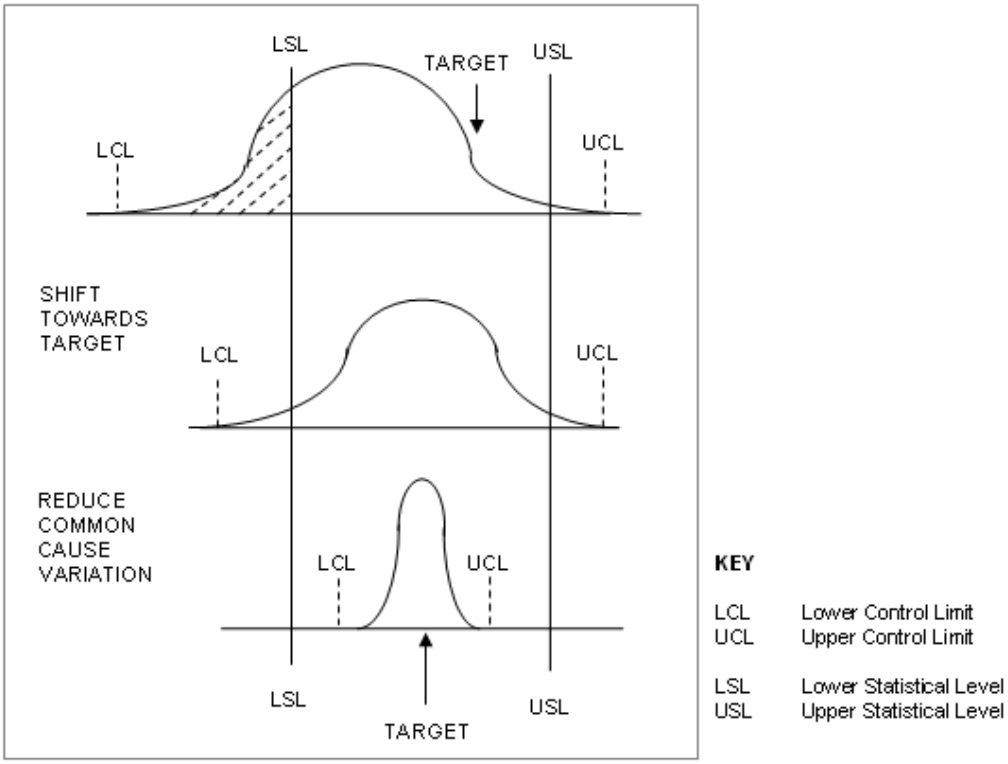


Figure 1-9 Making a Process Capable

Deming defines tampering as "action taken on a stable system in response to variation within statistical control, in an effort to compensate for this variation – the results of which will inevitably increase the variation and will increase cost from here on out." Tampering is any adjustment to a process (typically by operator or machine) in response to variation due to common causes (i.e., that variation between the control limits). By definition, process variation (due to common causes) is expected and is not a reason for adjusting or changing the process (tampering). Management that does not understand variation, time and time again asks for an explanation or corrective action when confronted with variation due to common causes.

1.2.4.2 Do Testers Need to Know SPC?

Testing is a measurement process. It attempts to measure the implemented software against either or both specifications and user needs. Statistical process control is a measurement tool.

The more you know about the process used to develop software, the more effective the testing process will become. For example, if you know that the requirements process has significant variability, meaning there is a high probability that the requirements as defined are not correct, you should then focus testing efforts on determining the "correctness" of the requirements as viewed by the customer. Software testing does not add a lot of value to the business if all they are doing is validating that incorrect requirements are implemented correctly.

1.2.5 Software Product Defects

The U.S. General Accounting Office summarized the errors detected in computerized applications they reviewed. It is reasonable to assume that these defects are typical of most computer systems and thus those problems should be included in any test program. These problems, resulting in the applications automatically initiating uneconomical or otherwise incorrect actions, can be broadly categorized as software design defects and data defects.

1.2.5.1 Software Design Defects

Software design defects that most commonly cause bad decisions by automated decision-making applications include:

- Designing software with incomplete or erroneous decision-making criteria. Actions have been incorrect because the decision-making logic omitted factors that should have been included. In other cases, decision-making criteria included in the software were appropriate, either at the time of design or later, because of changed circumstances.
- Failing to program the software as intended by the customer (user), or designer, resulting in logic errors often referred to as programming errors.
- Omitting needed edit checks for determining completeness of output data. Critical data elements have been left blank on many input documents, and because no checks were included, the applications processed the transactions with incomplete data.

1.2.5.2 Data Defects

Input data is frequently a problem. Since much of this data is an integral part of the decision-making process, its poor quality can adversely affect the computer-directed actions. Common problems are:

- Incomplete data used by automated decision-making applications. Some input documents prepared by people omitted entries in data elements that were critical to the application but were processed anyway. The documents were not rejected when incomplete data was being used. In other instances, data needed by the application that should have become part of IT files was not put into the system.
- Incorrect data used in automated decision-making application processing. People have often unintentionally introduced incorrect data into the IT system.
- Obsolete data used in automated decision-making application processing. Data in the IT files became obsolete due to new circumstances. The new data may have been available but was not put into the computer.

1.2.6 Finding Defects

All testing focuses on discovering and eliminating defects or variances from what is expected. Testers need to identify these two types of defects:

- Variance from Specifications – A defect from the perspective of the builder of the product.
- Variance from what is Desired – A defect from a user (or customer) perspective.

Typical software system defects include:

- IT improperly interprets requirements

  IT staff misinterprets what the user wants, but correctly implements what the IT people believe is wanted.

- Users specify the wrong requirements

  The specifications given to IT are erroneous.

- Requirements are incorrectly recorded

  IT fails to record the specifications properly.

- Design specifications are incorrect

  The application system design does not achieve the system requirements, but the design as specified is implemented correctly.

- Program specifications are incorrect

  The design specifications are incorrectly interpreted, making the program specifications inaccurate; however, it is possible to properly code the program to achieve the specifications.

- Errors in program coding

  The program is not coded according to the program specifications.

- Data entry errors

  Data entry staff incorrectly enters information into your computers.

- Testing errors

  Tests either falsely detect an error or fail to detect one.

- Mistakes in error correction

  Your implementation team makes errors in implementing your solutions.

- The corrected condition causes another defect

In the process of correcting a defect, the correction process itself injects additional defects into the application system.

1.3 Reducing the Frequency of Defects in Software Development

In the early days of computing, experience showed that some software development processes were much more effective than other software development processes. As one of the major purchasers of customized software, the Department of Defense (DOD) undertook a study to identify the criteria that made software development more effective. This research was conducted by Carnegie Mellon University's Software Engineering Institute (SEI). The end result was an algorithm that enabled the DOD to identify the more effective and efficient software development. This algorithm is now known as the capability maturity model.

1.3.1 The Five Levels of Maturity

QAI follows SEI's maturity model with minimal changes. Figure 1-10 illustrates SE's five levels of maturity in a quality management environment.

What the capability maturity model does is identify five different levels of maturity. As the model moves from Level 1 to Level 5, the variability in the process is significantly reduced. Thus, those at Level 5 have minimal variability in their software development process, while Level 1 organizations have significant variability. The cost differences to produce a function point of logic between a Level 1 and Level 5 organization may vary by 100 times. In other words, what a Level 1 organization may spend on building software, for example $1,000, may only cost $10 for a Level 5 organization.

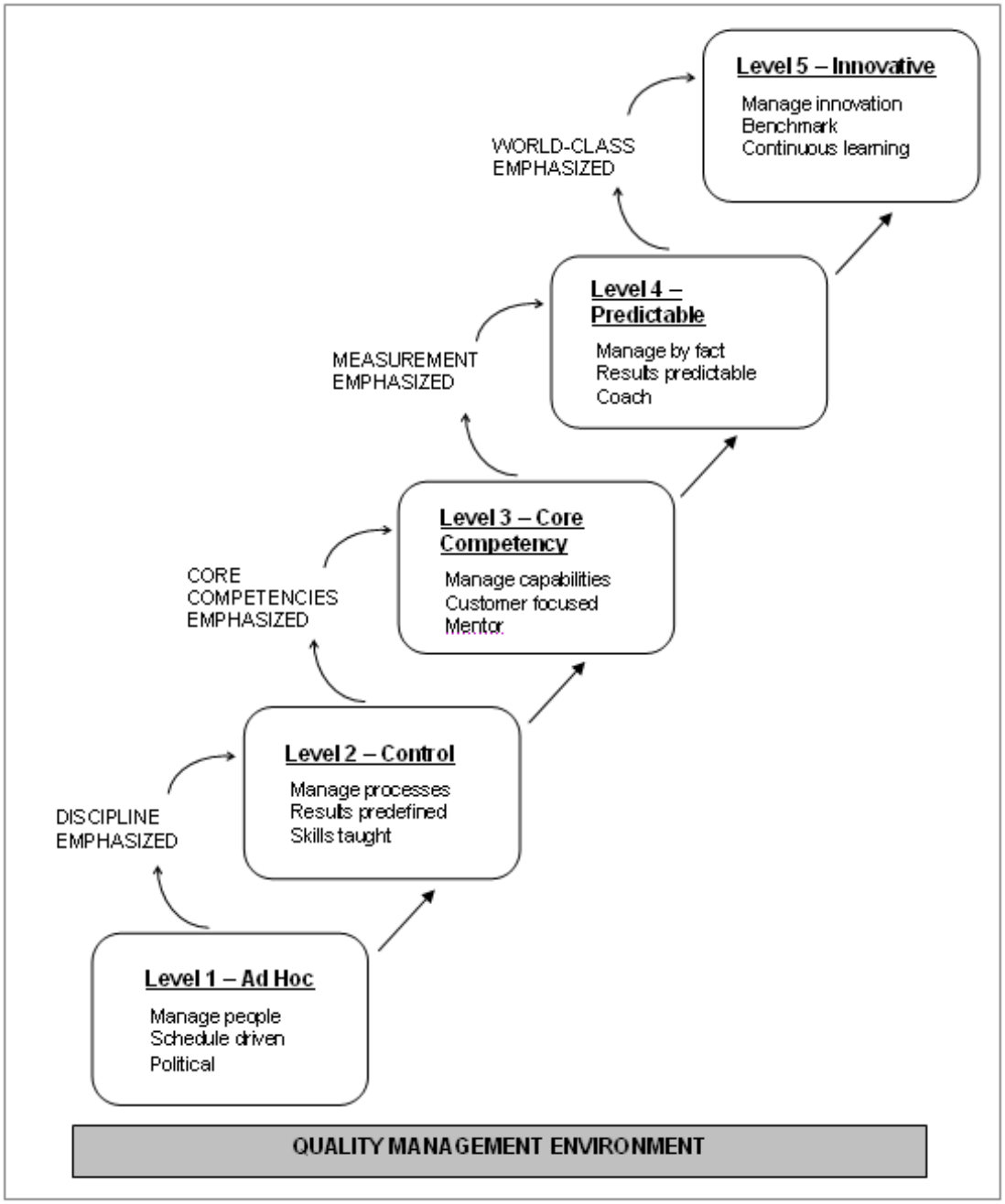


Figure 1-10 The Five Levels of Process Maturity

1.3.1.1 Level 1 – Ad Hoc

Ad hoc means unstructured, inconsistent levels of performance. At the ad hoc level, tasks are not performed the same way by different people or different groups. For example, one system development group may use part of the system development methodology, but improvise

other parts; another group may select different parts of the same system development methodology to use, and decide not to perform tasks done by a previous group.

At this level, management manages people and jobs. Management will establish goals or objectives for individuals and teams, and manage to those objectives and goals with minimal concern about the means used to achieve the goals. This level is normally heavily schedule driven, and those that meet the schedules are rewarded. Since there are not standards against which to measure deliverables, people's performance is often dependent upon their ability to convince management that the job they have done is excellent. This causes the environment to be very political. Both management and staff become more concerned with their personal agenda than with meeting their organization's mission.

The emphasis needed to move from Level 1 to Level 2 is discipline and control. The emphasis is on getting the work processes defined, training the people in the work processes, implementing sufficient controls to assure compliance to the work processes, and producing products that meet predefined standards.

1.3.1.2 Level 2 – Control

There are two major objectives to be achieved at Level 2. The first is to instill discipline in the culture of the information organization so that through the infrastructure, training, and leadership of management individuals will want to follow defined processes. The second objective is to reduce variability in the processes by defining them to a level that permits relatively constant outputs. At this level, processes are defined with minimal regard to skills needed to perform the process AND with minimal regard to the impact on other processes. At Level 2, the work processes are defined; management manages those processes, and uses validation and verification techniques to check compliance to work procedures and product standards. Having the results predefined through a set of standards enables management to measure people's performance against meeting those standards. Education and training are an important component of Level 2, as is building an infrastructure that involves the entire staff in building and improving work processes.

The emphasis that needs to be put into place to move to Level 3 is defining and building the information group's core competencies.

1.3.1.3 Level 3 – Core Competency

At this level, an information organization defines its core competencies and then builds an organization that is capable of performing those core competencies effectively and efficiently. The more common core competencies for an information services organization include system development, maintenance, testing, training, outsourcing, and operation. The information group must decide if it wants core competencies in fields such as communication, hardware and software selection, contracting, and so forth. Once the core competencies are determined, then the processes defined at Level 2 must be reengineered to drive the core competencies. In addition, the tasks are analyzed to determine what skills are needed to perform those processes. Next, a staff must be retrained, recruited, motivated, and supported to perform

those core competencies in an effective and efficient manner. It is the integration of people and processes, coupled with managers with people management skills, which are needed to maintain and improve those core competencies. Lots of mentoring occurs at this level, with the more experienced people building skills in the less experienced. It is also a level that is truly customer focused – both the information organization and the customer know the information group's core competencies.

The managerial emphasis that is needed to move to Level 4 is quantitative measurement. Measurement is only a practical initiative when the processes are stabilized and focused on achieving management's desired results.

1.3.1.4 Level 4 – Predictable

This level has two objectives. The first is to develop quantitative standards for the work processes based on performance of the Level 3 stabilized processes. The second objective is to provide management the dashboards and skill sets needed to manage quantitatively. The result is predictable work processes. Knowing the normal performance of a work process, management can easily identify problems through variation from the quantitative standards to address problems quickly to keep projects on schedule and budget. This level of predictability is one that uses measurement to manage as opposed to using measurement to evaluate individual performance. At this level, management can become coaches to help people address their day-to-day challenges in performing work processes in a predictable manner. Management recognizes that obstacles and problems are normal in professional activities, and through early identification and resolution, professional work processes can be as predictable as manufacturing work processes.

The management emphasis that is needed to move to Level 5 is one of desiring to be world-class. World-class means doing the best that is possible, given today's technology.

1.3.1.5 Level 5 – Innovative

At Level 5, the information organization wants to be a true leader in the industry. At this level, the organization is looking to measure itself against the industry through benchmarking, and then define innovative ways to achieve higher levels of performance. Innovative approaches can be achieved through benchmarking other industries, applying new technology in an innovative way, reengineering the way work is done, and by constantly studying the literature and using experts to identify potential innovations. This level is one in which continuous learning occurs, both in individuals and the organization.

1.3.2 Testers Need to Understand Process Maturity

Software testers face a much greater challenge testing software developed by maturity Level 1, than they do by testing software developed by higher maturity levels. Some have categorized Level 1 organizations as "Test and Fix" organizations. At this level, testing and

rework will consume more than 50% of the total software development effort. As software development processes mature, two things happen: more testing occurs during the building of software and the amount of testing required is reduced.

1.4 Factors Affecting Software Testing

Software testing varies from organization to organization. Many factors affect testing. The major factors are:

- People relationships
- Scope of testing
- Misunderstanding life cycle testing
- Poorly developed test plans
- Testing constraints

Each of these factors will be discussed individually to explain how the role of testing in an IT organization is determined.

1.4.1 People Relationships

With the introduction of the CSTE certification, testing is finally being recognized as a profession with a specialized skill set and qualifications. Organizations are finally becoming convinced that testers truly can not test in quality at the end of the development process and that the traditional waterfall methodology has led to many of the issues surrounding testing today. We now understand that testing has typically not been well defined and leaving it as the last activity in the development process was not the best approach.

The word “testing” conjures up a variety of meanings depending upon an individual’s frame of reference. Some people view testing as a method or process by which they add value to the development cycle; they can even enjoy the challenges and creativity of testing. Other people feel that testing tries a person’s patience, fairness, ambition, credibility, and capability. Testing can actually affect a person’s mental and emotional health if you consider the office politics and interpersonal conflicts that are often times present.

Some attitudes that have shaped a negative view of testing and testers are:

- Testers hold up implementation.
- Giving testers less time to test will reduce the chance that they will find defects.
- Letting the testers find problems is an appropriate way to debug.
- Defects found in production are the fault of the testers.
- Testers do not need training; only programmers need training.

Although testing is a process, it is very much a dynamic one in that the process will change somewhat with each application under test. There are several variables that affect the testing process including: the development process itself, software risk, customer/user participation, the testing process, a tester's skill set, use of tools, testing budget and resource constraints, management support, and morale and motivation of the testers. It is obvious that the people side of software testing has long been ignored for the more process-related issues of test planning, test tools, defect tracking, and so on.

According to the book, Surviving the Top Ten Challenges of Software Testing, A People-Oriented Approach by William Perry and Randall Rice, the top ten people challenges have been identified as:

- Training in testing
- Relationship building with developers
- Using tools
- Getting managers to understand testing
- Communicating with users about testing
- Making the necessary time for testing
- Testing "over the wall" software
- Trying to hit a moving target
- Fighting a lose-lose situation
- Having to say "no"

Testers should perform a self-assessment to identify their own strengths and weaknesses as they relate to people-oriented skills. They should also learn how to improve the identified weaknesses, and build a master plan of action for future improvement.

Essential testing skills include test planning, using test tools (automated and manual), executing tests, managing defects, risk analysis, test measurement, designing a test environment, and designing effective test cases. Additionally, a solid vocabulary of testing is essential. A tester needs to understand what to test, who performs what type of test, when testing should be performed, how to actually perform the test, and when to stop testing.

1.4.2 Scope of Testing

The scope of testing is the extensiveness of the test process. A narrow scope may be limited to determining whether or not the software specifications were correctly implemented. The scope broadens as more responsibilities are assigned to software testers.

Among the broader scope of software testing are these responsibilities:

1. Software testing can compensate for the fact that the software development process does not identify the true needs of the user, and thus test to determine whether or not the user's needs have been met regardless of the specifications.

2. Finding defects early in the software development process when they can be corrected at significantly less cost than detected later in the software development process.

3. Removing defects of all types prior to software going into a production state when it is significantly cheaper than during operation.

4. Identifying weaknesses in the software development process so that those processes can be improved and thus mature the software development process. Mature processes produce software more effectively and efficiently.

In defining the scope of software testing each IT organization must answer the question, “Why are we testing?”

1.4.3 Misunderstanding Life Cycle Testing

The traditional view of the development life cycle places testing prior to operation and maintenance as illustrated in Table 1-5. All too often, testing after coding is the only method used to determine the adequacy of the system. When testing is constrained to a single phase and confined to the later stages of development, severe consequences can develop. It is not unusual to hear of testing consuming 50 percent of the development budget. All errors are costly, but the later in the life cycle that the error discovered is made, the more costly the error. An error discovered in the latter parts of the life cycle must be paid for four different times. The first cost is developing the program erroneously, which may include writing the wrong specifications, coding the system wrong, and documenting the system improperly. Second, the system must be tested to detect the error. Third, the wrong specifications and coding must be removed and the proper specifications, coding, and documentation added. Fourth, the system must be retested to determine that it is now correct.

Table 1-5: Life Cycle Testing Activities

Life Cycle Phase	Testing Activities
Requirements	- Determine verification approach - Determine adequacy of design - Generate functional test data - Determine consistency of design with requirements
Design	- Determine adequacy of design - Generate structural and functional test data - Determine consistency with design
Program (build/construction)	- Determine adequacy of implementation - Generate structural and functional test data for programs
Test	- Test application system
Installation	- Place tested system into production
Maintenance	- Modify and retest

If lower cost and higher quality systems are the information services goals, verification must not be isolated to a single phase in the development process, but rather, incorporated into each phase of development. One of the most prevalent and costly mistakes on systems development projects today is to defer the activity of detecting and correcting problems until late in the project. A major justification for an early verification activity is that many costly errors are made before coding begins.

Studies have shown that the majority of system errors occur in the design phase. These numerous studies show that approximately two-thirds of all detected system errors can be attributed to errors made during the design phase. This means that almost two-thirds of the errors must be specified and coded into programs before they can be detected. The recommended testing process is presented in Table 1-5 as a life cycle chart showing the verification activities for each phase. The success of conducting verification throughout the development cycle depends upon the existence of clearly defined and stated products at each development stage. The more formal and precise the statement of the development product, the more amenable it is to the analysis required to support verification. Many of the new system development methodologies encourage firm products even in the early development stages.

The recommended test process involves testing in every phase of the life cycle. During the requirements phase, the emphasis is upon validation to determine that the defined requirements meet the needs of the organization. During the design and program phases, the emphasis is on verification to ensure that the design and programs accomplish the defined requirements. During the test and installation phases, the emphasis is on inspection to determine that the implemented system meets the system specification. During the maintenance phases, the system will be retested to determine that the changes work and that the unchanged portion continues to work.

The following activities should be performed at each phase of the life cycle:

- Analyze the structures produced at this phase for internal testability and adequacy.
- Generate test sets based on the structure at this phase.

In addition, the following should be performed during design and programming:

- Determine that the structures are consistent with structures produced during previous phases.
- Refine or redefine test sets generated earlier.

Throughout the entire life cycle, neither development nor verification is a straight-line activity. Modifications or corrections to a structure at one phase will require modifications or re-verification of structures produced during previous phases.

1.4.3.1 Requirements

The verification activities that accompany the problem definition and requirements analysis phase of software development are extremely significant. The adequacy of the requirements must be thoroughly analyzed and initial test cases generated with the expected (correct) responses. Developing scenarios of expected system use helps to determine the test data and anticipated results. These tests form the core of the final test set. Generating these tests and the expected behavior of the system clarifies the requirements and helps guarantee that they are testable. Vague requirements or requirements that are not testable leave the validity of the delivered product in doubt. Late discovery of requirements inadequacy can be very costly. A determination of the criticality of software quality attributes and the importance of validation should be made at this stage. Both product requirements and validation requirements should be established.

1.4.3.2 Design

Organization of the verification effort and test management activities should be closely integrated with preliminary design. The general testing strategy – including test methods and test evaluation criteria – is formulated, and a test plan is produced. If the project size or criticality warrants, an independent test team is organized. In addition, a test schedule with observable milestones is constructed. At this same time, the framework for quality assurance and test documentation should be established.

During detailed design, validation support tools should be acquired or developed and the test procedures themselves should be produced. Test data to exercise the functions introduced during the design process, as well as test cases based upon the structure of the system, should be generated. Thus, as the software development proceeds, a more effective set of test cases is built.

In addition to test organization and the generation of test cases, the design itself should be analyzed and examined for errors. Simulation can be used to verify properties of the system structures and subsystem interaction; the developers should verify the flow and logical structure of the system, while the test team should perform design inspections using design walkthroughs. Missing cases, faulty logic, module interface mismatches, data structure

inconsistencies, erroneous I/O assumptions, and user interface inadequacies, are items of concern. The detailed design must prove to be internally coherent, complete, and consistent with the preliminary design and requirements.

1.4.3.3 Program (Build/Construction)

Actual testing occurs during the construction stage of development. Many testing tools and techniques exist for this stage of system development. Code walkthrough and code inspection are effective manual techniques. Static analysis techniques detect errors by analyzing program characteristics such as data flow and language construct usage. For programs of significant size, automated tools are required to perform this analysis. Dynamic analysis, performed as the code actually executes, is used to determine test coverage through various instrumentation techniques. Formal verification or proof techniques are used to provide further quality control.

1.4.3.4 Test Process

During the test process, careful control and management of test information is critical. Test sets, test results, and test reports should be catalogued and stored in a database. For all but very small systems, automated tools are required to do an adequate job – the bookkeeping chores alone become too large to handle manually. A test driver, test data generation aids, test coverage tools, test results management aids, and report generators are usually required.

1.4.3.5 Installation

The process of placing tested programs into production is an important phase normally executed within a narrow time span. Testing during this phase must ensure that the correct versions of the program are placed into production; that data if changed or added is correct; and that all involved parties know their new duties and can perform them correctly.

1.4.3.6 Maintenance

A large portion of the life cycle costs of a software system are spent on maintenance. As the system is used, it is modified either to correct errors or to augment the original system. After each modification the system must be retested. Such retesting activity is termed regression testing. The goal of regression testing is to minimize the cost of system revalidation. Usually only those portions of the system impacted by the modifications are retested. However, changes at any level may necessitate retesting, re-verifying and updating documentation at all levels below it. For example, a design change requires design re-verification, unit retesting and subsystem retesting. Test cases generated during system development are reused or used after appropriate modifications. The quality of the test documentation generated during system development and modified during maintenance will affect the cost of regression testing. If test data cases have been catalogued and preserved, duplication of effort will be minimized.

1.4.4 Poorly Developed Test Planning

It is widely agreed that variability in test planning is a major factor affecting software testing. A plan should be developed that defines how testing should be performed (see Skill Category 4 for building a test plan). With a test plan, testing can be considered complete when the plan has been accomplished. The test plan is a contract between the software stakeholders and the testers.

A typical test plan is illustrated in Figure 1-11. This plan will need to be customized for any specific software system. The applicable test objectives would be listed and ranked, and the phases of development would be listed as the phases in which testing must occur.

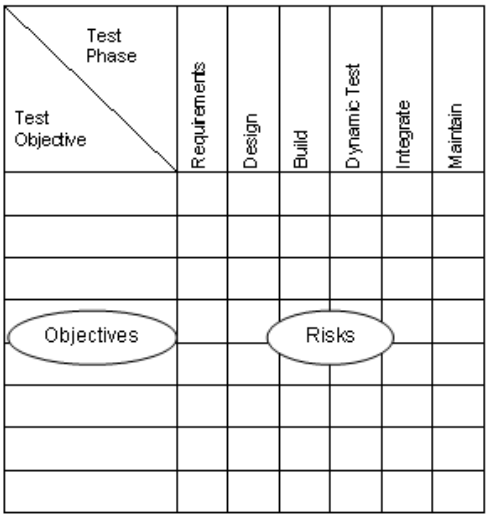


Figure 1-11 Example of a High-Level Test Plan

The following four steps must be followed to develop a customized test plan:

1. Select and rank test objectives.

   The customers or key users of the system in conjunction with the test team should define and rank the test objectives. In most instances, a limited number of test objectives will be defined. Statistically, if the major test objectives are defined and ranked, potential other test objectives will normally be addressed in a manner consistent with achieving the major test objectives. These should be listed in the matrix in sequence from the most significant objective to the least significant.

2. Identify the system development phases.

   The project development team should identify the phases of their development process. This is normally obtained from the system development methodology. These phases should be recorded in the test phase component of the matrix.

3. Identify the business risks associated with the system under development.

   The risks are the reasons the test objectives might not be met. The developers, key users, customers, and test personnel should brainstorm the risks associated with the software system. Most organizations have a brainstorming technique and it is appropriate for

individuals to use the technique in which they have had training and prior use. Using this technique, the risks should be identified and agreed upon by the group. The risks should then be ranked into high, medium, and low. This is a relational severity indicator, meaning that one-third of all risks should be indicated as high; one-third, medium; and one-third, low.

4. Place risks in the matrix.

   The risk team should determine the test phase in which the risk needs to be addressed by the test team and the test objective to which the risk is associated. Take the example of a payroll system. If there were a concern about compliance to federal and state payroll laws, the risk would be the penalties associated with noncompliance. Assuming assuring compliance to payroll laws was picked as one of the significant test objectives, the risk would be most prevalent during the requirements phase. Thus, in the matrix at the intersection between the compliance test objective and the requirements phase, the risk of "penalties associated with noncompliance to federal and state payroll laws" should be inserted. Note that a reference number, cross-referencing the risk, may do this. The risk would then have associated with it an H, M, or L, for high, medium, or low risk.

The completed matrix is a pictorial representation of a test plan.

1.4.5 Testing Constraints

Anything that inhibited the tester's ability to fulfill their responsibilities is a constraint. Constraints include:

- Limited schedule and budget
- Lacking or poorly written requirements
- Changes in technology
- Limited tester skills

Each of these four constraints will be discussed individually.

1.4.5.1 Budget and Schedule Constraints

Budget and schedule constraints may limit the ability of a tester to complete their test plan. Understanding why the lack of life cycle testing can cause budget and schedule problems can help relieve the constraint.

The cost of defect identification and correction increases exponentially as the project progresses. Figure 1-12 illustrates the accepted industry standard for estimating costs, and shows how costs dramatically increase the later you find a defect. A defect encountered during requirement and design is the cheapest to fix. So let's say it costs x; based on this a defect corrected during the system test phase costs 10x to fix. A defect corrected after the system goes into production costs 100x. Clearly, identifying and correcting defects early is the most cost-effective way to develop an error-free system.

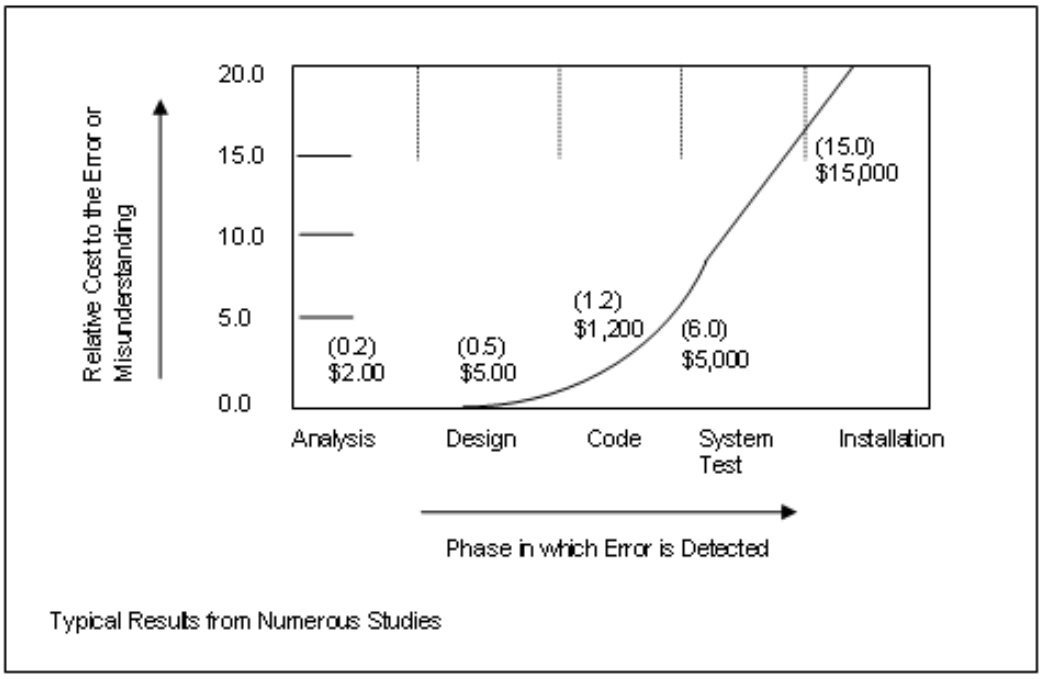


Figure 1-12 Relative Cost versus the Project Phase

Testing should begin during the first phase of the life cycle and continue throughout the life cycle. It's important to recognize that life cycle testing is essential to reducing the cost of testing.

Let's look at the economics of testing. One information services manager described testing in the following manner. "Too little testing is a crime – too much testing is a sin." The risk of under testing is directly translated into system defects present in the production environment. The risk of over testing is the unnecessary use of valuable resources in testing computer systems that have no flaws, or so few flaws that the cost of testing far exceeds the value of detecting the system defects. When control is viewed as a risk situation, this can result in over-and-under testing.

Most problems associated with testing occur from one of the following causes:

- Failing to define testing objectives
- Testing at the wrong phase in the life cycle
- Using ineffective test techniques

The cost-effectiveness of testing is illustrated in Figure 1-13. As the cost of testing increases, the number of undetected defects decreases. The left side of the illustration represents an under test situation in which the cost of testing is less than the resultant loss from undetected defects.

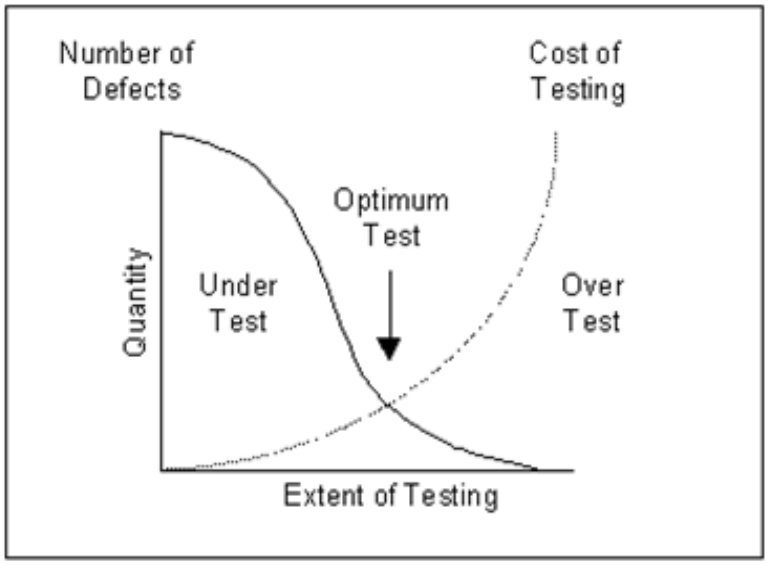


Figure 1-13 Testing Cost Curve

At some point, the two lines cross and an over test condition begins. In this situation, the cost of testing to uncover defects exceeds the losses from those defects. A cost-effective perspective means testing until the optimum point is reached, which is the point where the cost of testing no longer exceeds the value received from the defects uncovered.

Few organizations have established a basis to measure the effectiveness of testing. This makes it difficult for the individual systems analyst/programmer to determine the cost effectiveness of testing. Without testing standards, the effectiveness of the process cannot be evaluated in sufficient detail to enable the process to be measured and improved.

The use of standardized testing methodology provides the opportunity for a cause and effect relationship to be determined. In other words, the effect of a change in the methodology can be evaluated to determine whether that effect resulted in a smaller or larger number of defects. The establishment of this relationship is an essential step in improving the test process.

The cost-effectiveness of a testing process can only be determined when the effect of that process can be measured. When the process can be measured, it can be adjusted to improve the cost-effectiveness of the test process for the organization.

1.4.5.2 Lacking or Poorly Written Requirements

If requirements are lacking or poorly written, then the test team must have a defined method for uncovering and defining test objectives.

A test objective is simply a testing “goal.” It is a statement of what the test team or tester is expected to accomplish or validate during a specific testing activity. Test objectives, usually

defined by the test manager or test team leader during requirements analysis, guide the development of test cases, test scripts, and test data.

Test objectives enable the test manager and project manager to gauge testing progress and success, and enhance communication both within and outside the project team by defining the scope of the testing effort.

Each test objective should contain a statement of the objective, and a high-level description of the expected results stated in measurable terms. The users and project team must prioritize the test objectives. Usually the highest priority is assigned to objectives that validate high priority or high-risk requirements defined for the project. In cases where test time is cut short, test cases supporting the highest priority objectives would be executed first.

Test objectives can be easily derived from using the system requirements documentation, the test strategy, the outcome of the risk assessment, and the test team assignments. A few techniques for uncovering and defining test objectives if the requirements are poorly written include brainstorming and relating test objectives to the system inputs, events, or system outputs. Ideally, there should be less than 100 test objectives for all but the very largest systems. Test objectives are not simply a restatement of the system's requirements, but the actual way the system will be tested to assure that the system objective has been met. Completion criteria define the success measure for the tests.

As a final step, the test team should perform quality control. This activity entails using a checklist or worksheet to ensure that the process to set test objectives was followed, or reviewing them with the system users.

1.4.5.3 Changes in Technology

Effective testing must be done by a team comprised of information services professionals and users. In corporations where the users are not readily available, i.e., they are in a remote location, a professional test group can represent the users. Also vendors of software may not be able, or may not want to have users testing their systems during the developmental process. Again, in these instances, a professional test group can represent the users. The test group is known by different names, including IT testing, quality control, quality assurance, and inspectors.

The following technological developments are causing organizations to revise their approach to testing:

- Integration

  Technology is being more closely integrated into day-to-day business resulting in business being unable to operate without computer technology. For example, the airlines can only take reservations when their computer systems are operational.

- System Chains

  Computer systems are interconnected into cycles of chains such that problems in one can cascade into and affect others.

- The Domino Effect

  One problem condition, such as a wrong price or a program defect, can cause hundreds or even thousands of similar errors within a few minutes.

- Reliance on Electronic Evidence

  With hard-copy documents being removed from processing, the validity of the transactions is dependent upon the adequacy of controls, and thus a control error may result in extensive losses.

- Multiple Users

  Systems no longer belong to single users, but rather to multiple users, making it difficult to identify a single organizational unit responsible for a system.

The organizational approach to testing commences with a policy on testing computer systems. The policy should be developed under the direction of the IT department, but should represent the philosophy of the entire organization. Once the policy has been established, then the procedures and the methods of testing can be developed based upon the desires of management as expressed in the testing policy.

1.4.5.4 Limited Tester Skills

Testers should be competent in all ten of the CSTE Common Body of Knowledge skill categories to be effective. Lack of the skills needed for a specific test assignment constrains the ability of the testers to effectively complete that assignment.

1.5 Life Cycle Testing

Life cycle testing involves continuous testing of the solution even after software plans are complete and the tested system is implemented. At several points during the development process, the test team should test the system in order to identify defects at the earliest possible point.

Life cycle testing cannot occur until you formally develop your process. IT must provide and agree to a strict schedule for completing various phases of the process for proper life cycle testing to occur. If IT does not determine the order in which they deliver completed pieces of software, it’s impossible to schedule and conduct appropriate tests.

Life cycle testing is best accomplished by forming a test team. The team is composed of project members responsible for testing the system. They must use structured methodologies when testing; they should not use the same methodology for testing that they used for developing the system. The effectiveness of the test team depends on developing the system under one methodology and testing it under another. The life cycle testing concept is illustrated in Figure 1-14.

It shows that when the project starts, both the development process and system test process also begin. Thus, the testing and implementation teams begin their work at the same time and with the same information. The development team defines and documents the requirements for implementation purposes, and the test team uses those requirements for the purpose of testing the system. At appropriate points during the development process the test team runs the compliance process to uncover defects. The test team should use the structured testing techniques outlined in this book as a basis of evaluating the corrections.

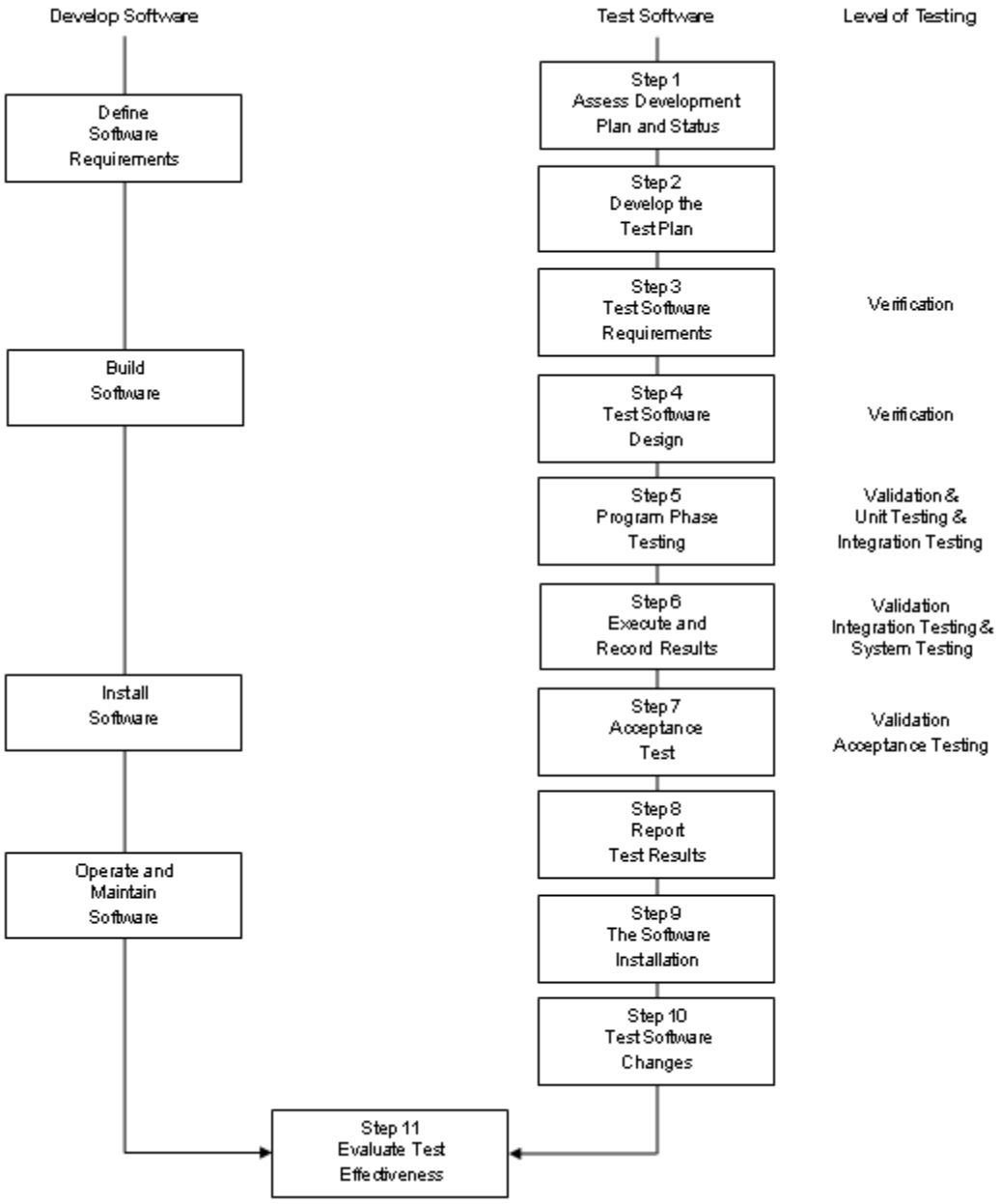


Figure 1-14 Life Cycle Testing

As you're testing the implementation, prepare a series of tests that your IT department can run periodically after your revised system goes live. Testing does not stop once you've completely implemented your system; it must continue until you replace or update it again!

1.6 Test Matrices

The test matrix shows the interrelationship between functional events and tests. The completed test matrix defines the conditions that must be tested during the test process to verify the proper functioning of the application system. It does not represent the totality of testing because there may be types of tests that verify the structure of the system, such as verifying the correct use of program statements that are not included in the test matrix.

The left side of the matrix shows the functional events and the top identifies the tests that occur on those events. Within the matrix cells are the process that needs to be tested. During requirements, the process may be generalized and vague, but it must become more specific as the life cycle progresses.

The example illustrated in Figure 1-15 is for the functional event of an employee getting a pay increase. The tests have been selected because each test:

- Represents the actions that must be performed on this functional event in order for the employee to get a raise
- Represents a task that is performed individually
- Can be associated with the individual responsible to perform that action
- Is broad enough to limit the actions to a reasonable number

	Actions								
Economic Events	**Initiate Event**	**Increase Approved**	**Data Entry**	**Form Storage**	**Data Entry Validation**	**Logical Validation**	**Update Pay Record**	**Audit Trail**	**Report**
Give Pay Increase	Supervisor completes Form X	Management initials Form X	Verify amount	Store Form X 90 days in Personnel	1. Numeric 2. Under $100	1.Employee exists 2. Within pay range 3. Within ± 15%	Change pay rate amount	Put change on payroll history file	Confirmation to supervisor
Event									
Event									

Figure 1-15 Testing Matrix

In the figure example there are nine identified conditions that must be tested to determine whether or not employees' pay increases are processed correctly by the application system.

Let's examine the nine tests to illustrate the conditions that must be tested:

- Initiate Event Action

  This action creates the functional event which will eventually give an employee a pay raise. Within the matrix is the process that needs to be evaluated. The process is for the supervisor to complete Form X. The information on the matrix must be condensed. If we were working with the application we would know, for example, that Form X was the organizational form for initiating a pay increase.

- Increase Approved Action

  After the supervisor has completed the form it must be approved by management. The test condition indicates that management will initial Form X. Our test process or test condition would then be designed to verify that procedures are established so that this will happen. Note that this and the previous action are manual actions.

- Data Entry Action

  The information on Form X is entered into the computer through a key entry process. The test condition is to verify that the keyed information is correct. Since this is in the early life cycle stages, we may not know all of the detailed information that will be keyed in or the verification process, but even in requirements we could substantiate that the requirements include a process to verify the accurate entry of data into the computer.

- Form Storage Action

  Form X needs to be retained until it is no longer of value to the organization. The test condition states that the form should be stored for 90 days in the personnel department. Again, our test process would verify that this is reasonable and that the system process provides for this condition.

- Data Entry Validation Action

  The verification that the information entered into the computer is correct. The test conditions outlined in the matrix state that the application system should validate that the pay increase amount is numeric, and that the increase is less than $100.

- Logical Validation Action

  The test suggests that additional validation determination will be made beyond the data values. Three test conditions are outlined as: 1) a logical check that the employee exists, 2) the pay increase will be within the pay range the employee is authorized, and 3) that the pay increase is within plus or minus 15 percent of the amount the employee was earning prior to the increase.

- Updated Pay Record Action

  The pay amount increase should be added to the employee's pay rate so that the pay record will properly reflect the new amount. The test condition is to ensure that the changed pay rate amount in the storage area is performed correctly.

- Audit Trail Action

  A record should be maintained on the increases provided the employee. The action shows that the increase in payroll should be maintained on a history file.

- Report Action

  The results of computer processing should be sent back to the supervisor as a confirmation process. The test condition is to verify that the system provides for such a confirmation.

This testing matrix would be typical of one prepared during the requirements phase. The functional events, as well as the actions, will be used throughout the systems development life cycle. However, the test conditions will become more specific as the life cycle progresses. For example, there is an action indicating that data will be entered into the computer. During requirements, how that will occur may not be known, so that the test condition must still be generalized as Figure 1-15 Testing Matrix shows. However, as the system progresses through the life cycle, the testing matrix becomes more specific. Eventually, the testing matrix will be expanded to the point where test transactions can be prepared from the matrix information.

1.6.1 Cascading Test Matrices

The tests that occur on functional events vary based on the events themselves. If generic actions are used, it may be possible to include several functional events in the same matrix. However, it is generally better to limit a matrix to a single functional event.

Including only one functional event on a matrix provides the following two advantages:

- Tests can be customized for specific functional events
- Tests on the functional events can be the creation of new functional events which show a relationship between the events.

One functional event leading to the creation of another and leading to another will cause several matrices to be prepared. Properly prepared, they will demonstrate the cascading events illustrating how one event can create another event which can create yet another event. An example of a cascading matrix is illustrated in Figure 1-16. This matrix is from an order entry billing system.

	Action			
Economic Event	Initiate Order	Accept Order	Enter Order	Approve Credit
Order Product				•

	Action			
Economic Event	Authorize Customer	Credit No Good	Credit Approved	Ship Product
Credit Approval				•

	Action			
Economic Event				
Ship Product				

Figure 1-16 Cascading Test Matrices

The first functional event is the order for a product placed by a customer. The type of actions that would occur on this is the initiation of the order, the acceptance of the order, the entry of the order, and the credit approval action. That action creates a new functional event which is the formal approval of customer credit. Figure 1-16 shows how the action in the first matrix cascades or points to the second matrix.

The tests for the functional event of approving credit involves such tasks as determining that the customer is an authorized customer, causing a decision about the customer's credit. If good action occurs, it creates a new functional event to ship a product. The figure shows how the action to ship a product creates a new functional event and a new matrix. This process would continue until the entire order entry billing application was complete.

In the creation of the test plan, IT people sometimes lose track of the interrelationship of functional events. The creation of the cascading test matrix reinforces the interrelationship of functional events and enables that aspect of systems to be better tested.

1.7 Independent Testing

The primary responsibility of individuals accountable for testing activities is to ensure that quality is measured accurately. Often, just knowing that the organization is measuring quality is enough to cause improvements in the applications being developed. In the loosest definition of independence, just having a tester or someone in the organization devoted to test activities is a form of independence.

The roles and reporting structure of test resources differs across and within organizations. These resources may be business or systems analysts assigned to perform testing activities, or

may be testers who report to the project manager. Ideally, the test resources will have a reporting structure independent from the group designing or developing the application in order to assure that the quality of the application is given as much consideration as the project budget and timeline.

Misconceptions abound regarding the skill set required to perform testing, including:

- Testing is easy
- Anyone can perform testing
- No training or prior experience is necessary

In truth, to test effectively, an individual must:

- Thoroughly understand the system
- Thoroughly understand the technology the system is being deployed upon (e.g., client/server or Internet technologies introduce their own challenges)
- Possess creativity, insight, and business knowledge
- Understand the development methodology used and the resulting artifacts

While much of this discussion focuses on the roles and responsibilities of an independent test team, it is important to note that the benefits of independent testing can be seen in the unit testing stage. Often, successful development teams will have a peer perform the unit testing on a program or class. Once a portion of the application is ready for integration testing, the same benefits can be achieved by having an independent person plan and coordinate the integration testing.

Where an independent test team exists, they are usually responsible for system testing, the oversight of acceptance testing, and providing an unbiased assessment of the quality of an application. The team may also support or participate in other phases of testing as well as executing special test types such as performance and load testing.

An independent test team is usually comprised of a test manager or team leader and a team of testers. The test manager should join the team no later than the start of the requirements definition stage. Key testers may also join the team at this stage on large projects to assist with test planning activities. Other testers can join later to assist with the creation of test cases and scripts, and right before system testing is scheduled to begin.

The test manager ensures that testing is performed, that it is documented, and that testing techniques are established and developed. They are responsible for ensuring that tests are designed and executed in a timely and productive manner, as well as:

- Test planning and estimation
- Designing the test strategy
- Reviewing analysis and design artifacts
- Chairing the Test Readiness Review
- Managing the test effort
- Overseeing acceptance tests

Testers are usually responsible for:

- Developing test cases and procedures
- Test data planning, capture, and conditioning
- Reviewing analysis and design artifacts
- Testing execution
- Utilizing automated test tools for regression testing
- Preparing test documentation
- Defect tracking and reporting

Other testers joining the team will primarily focus on test execution, defect reporting, and regression testing. These testers may be junior members of the test team, users, marketing or product representatives, and so on.

The test team should be represented in all key requirements and design meetings including:

- JAD or requirements definition sessions
- Risk analysis sessions
- Prototype review sessions

They should also participate in all inspections or walkthroughs for requirements and design artifacts.

1.8 Tester's Workbench

The tester's workbench is a pictorial representation of how a specific test task is performed. Prior to understanding a tester's workbench it is important to understand a work process.

1.8.1 What is a Process?

A process can be defined as a set of activities that represent the way work is performed. The outcome from a process is usually a product or service. Both software development and software testing are processes. Table 1-6 illustrates a few examples of processes and their outcomes.

Table 1-6: Process Example and Outcomes

Examples of IT Processes	Outcomes
Analyze Business Needs	Needs Statement
Conduct JAD Session	JAD Notes
Run Job	Executed Job
Develop Strategic Plan	Strategic Plan
Recognize Individual Performance	Recognized Individual
Unit Test	Defect-free Unit

There are two ways to visually portray a process. One is the Plan Do Check Act (PDCA) cycle. The other is a workbench. The PDCA cycle is a conceptual view of a process, while the workbench is a more practical illustration of a process. Let's look at both views of software testing.

1.8.1.1 The PDCA View of a Process

Brief descriptions of the four components of the PDCA concept are provided below and are illustrated in Figure 1-17.

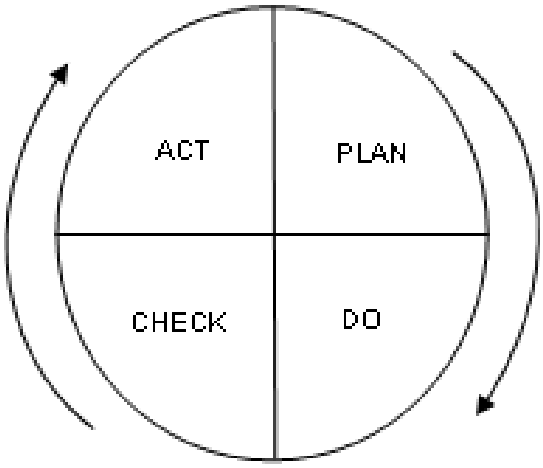


Figure 1-17 PDCA Concept

1.8.1.1.1 P – Plan - Devise Your Plan

Define your objective and determine the conditions and methods required to achieve your objective. Describe clearly the goals and policies needed to achieve the objective at this stage. Express a specific objective numerically. Determine the procedures and conditions for the means and methods you will use to achieve the objective.

1.8.1.1.2 D – Do (or Execute) the Plan

Create the conditions and perform the necessary teaching and training to execute the plan. Make sure everyone thoroughly understands the objectives and the plan. Teach workers the procedures and skills they need to fulfill the plan and thoroughly understand the job. Then perform the work according to these procedures.

1.8.1.1.3 C – Check the Results

Check to determine whether work is progressing according to the plan and whether the expected results are obtained. Check for performance of the set procedures, changes in conditions, or abnormalities that may appear. As often as possible, compare the results of the work with the objectives.

1.8.1.1.4 A – Act - Take the Necessary Action

If your checkup reveals that the work is not being performed according to plan or that results are not as anticipated, devise measures for appropriate action.

If a check detects an abnormality – that is, if the actual value differs from the target value – search for the cause of the abnormality and eliminate the cause. This will prevent the recurrence of the defect. Usually you will need to retrain workers and revise procedures to eliminate the cause of a defect.

1.8.1.2 The Workbench View of a Process

A process can be viewed as one or more workbenches, as shown in Figure 1-18. Each workbench is built on the following two components:

- Objective – States why the process exists, or its purpose.

  Example: A JAD session is conducted to uncover the majority of customer requirements early and efficiently, and to ensure that all involved parties interpret these requirements consistently.
- People Skills – The roles, responsibilities, and associated skill sets needed to execute a process. Major roles include suppliers, owners, and customers.

Each workbench has the following components:

- Inputs – The entrance criteria or deliverables needed to perform testing.
- Procedures – Describe how work must be done; how methods, tools, techniques, and people are applied to perform a process. There are Do procedures and Check procedures. Procedures indicate the “best way” to meet standards.

  Example*:*

  1. Scribe: Use the XYZ tool to enter the requirements. Generate a list of any open issues using the XX template.

  2. Leader: Walk through the requirements, paraphrasing each item. Address each open issue when its REF column contains the item being covered.

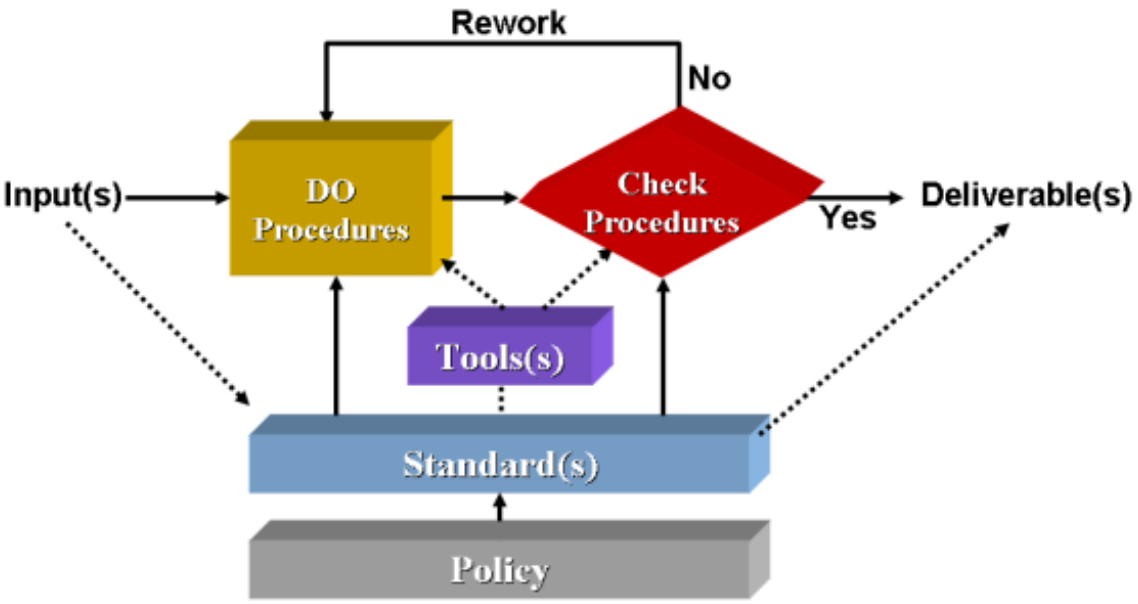


Figure 1-18 Workbench Concept

- Deliverables – Any product or service produced by a process. Deliverables can be interim or external (or major). Interim deliverables are produced within the workbench, but never passed on to another workbench. External deliverables may be used by one or more workbench, and have one or more customers. Deliverables serve as both inputs to and outputs from a process.

Example*:* JAD Notes are interim and Requirements Specifications are external.

- Standards – Measures used to evaluate products and identify nonconformance. The basis upon which adherence to policies is measured.
- Tools – Aids to performing the process.

Example*:* CASE tools, checklists, templates, etc.

1.8.1.3 Workbenches are Incorporated into a Process

To understand the testing process, it is necessary to understand the workbench concept. In IT, workbenches are more frequently referred to as phases, steps, or tasks. The workbench is a way of illustrating and documenting how a specific activity is to be performed. Defining workbenches is normally the responsibility of a Process Management Committee, or also known as a Standards Committee.

The workbench and the software testing process, which is comprised of many workbenches, are illustrated in Figure 1-19.

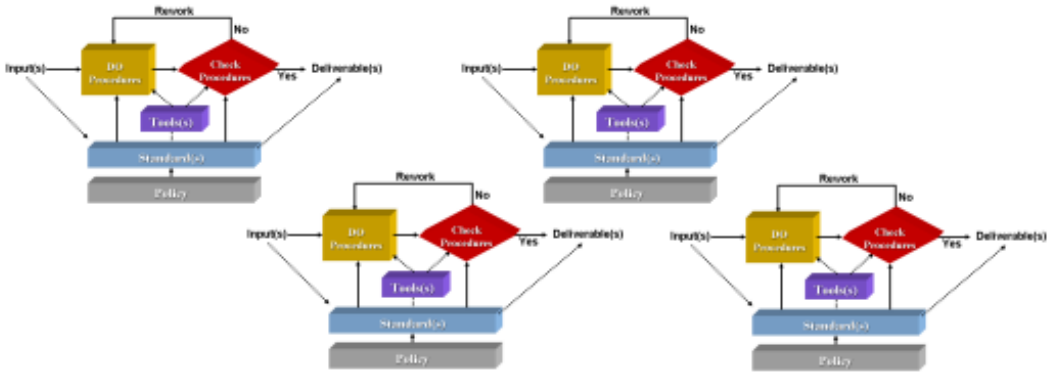


Figure 1-19 Software Testing Process

The workbench concept can be used to illustrate one of the steps involved in testing software systems. The tester's unit test workbench consists of these steps:

1. Input products (unit test specifications) are given to the tester.
2. Work is performed (e.g., debugging); a procedure is followed; a product or interim deliverable is produced, such as a defect list.
3. Work is checked to ensure the unit meets specs and standards, and that the procedure was followed.
4. If check finds no problems, product is released to the next workbench (e.g., integration testing).
5. If check finds problems, product is sent back for rework.

1.9 Levels of Testing

The sequence in which testing occurs is represented by different levels or types of testing. This sequence of testing is:

- Verification Testing

  Static testing of development interim deliverables

- Unit Testing

  These tests verify that the system functions properly; for example, pressing a function key to complete an action.

- Integration Testing

  The system runs tasks that involve more than one application or database to verify that it performed the tasks accurately.

- System Testing

  These tests simulate operation of the entire system, and verify that it ran correctly.

- User Acceptance Testing

  This real-world test means the most to your business; and, unfortunately, there's no way to conduct it in isolation. Once your organization staff, customers, or vendors begin to interact with your system, they'll verify that it functions properly for you.

The "V" Concept of Testing illustrates the sequence in which testing should occur.

1.9.1 Verification versus Validation

Verification ensures that the system (software, hardware, documentation, and personnel) complies with an organization's standards and processes, relying on review or non-executable methods. Validation physically ensures that the system operates according to plan by executing the system functions through a series of tests that can be observed and evaluated. Verification answers the question, "Did we build the right system?" while validations addresses, "Did we build the system right?"

Keep in mind that verification and validation techniques can be applied to every element of the computerized system. You'll find these techniques in publications dealing with the design and implementation of user manuals and training courses, as well as in industry publications.

1.9.1.1 Computer System Verification and Validation Examples

Verification requires several types of reviews, including requirements reviews, design reviews, code walkthroughs, code inspections, and test reviews. The system user should be involved in these reviews to find defects before they are built into the system. In the case of purchased systems, user input is needed to assure that the supplier makes the appropriate tests to eliminate defects. Table 1-7 shows examples of verification. The list is not exhaustive, but it does show who performs the task and what the deliverables are. For purchased systems, the term "developers" applies to the supplier's development staff.

Table 1-7: Computer System Verification Examples

Verification Example	Performed by	Explanation	Deliverable
Requirements Reviews	Developers, Users	The study and discussion of the computer system requirements to ensure they meet stated user needs and are feasible.	Reviewed statement of requirements, ready to be translated into system design.
Design Reviews	Developers	The study and discussion of the computer system design to ensure it will support the system requirements.	System design, ready to be translated into computer programs, hardware configurations, documentation, and training.
Code Walkthroughs	Developers	An informal analysis of the program source code to find defects and verify coding techniques.	Computer software ready for testing or more detailed inspections by the developer.
Code Inspections	Developers	A formal analysis of the program source code to find defects as defined by meeting computer system design specifications. Usually performed by a team composed of developers and subject matter experts.	Computer software ready for testing by the developer.

Validation is accomplished simply by executing a real-life function (if you wanted to check to see if your mechanic had fixed the starter on your car, you'd try to start the car). Examples of validation are shown in Table 1-8. As in the table above, the list is not exhaustive.

Table 1-8: Computer System Validation Examples

Validation Example	Performed by	Explanation	Deliverable
Unit Testing	Developers	The testing of a single program, module, or unit of code. Usually performed by the developer of the unit. Validates that the software performs as designed.	Software unit ready for testing with other system components, such as other software units, hardware, documentation, or users.
Integrated Testing	Developers with support from an independent test team	The testing of related programs, modules, or units of code. Validates that multiple parts of the system interact according to the system design.	Portions of the system ready for testing with other portions of the system.
System Testing	Independent Test Team	The testing of an entire computer system. This kind of testing can include functional and structural testing, such as stress testing. Validates the system requirements.	A tested computer system, based on what was specified to be developed or purchased.
User Acceptance Testing	Users with support from an independent test team	The testing of a computer system or parts of a computer system to make sure it will work in the system regardless of what the system requirements indicate.	A tested computer system, based on user needs.

Determining when to perform verification and validation relates to the development, acquisition, and maintenance of software. For software testing, this relationship is especially critical because:

- The corrections will probably be made using the same process for developing the software. If the software was developed internally using a waterfall methodology, that methodology will probably be followed in making the corrections; on the other hand, if the software was purchased or contracted, the supplier will likely make the correction. You'll need to prepare tests for either eventuality.
- Testers can probably use the same test plans and test data prepared for testing the original software. If testers prepared effective test plans and created extensive test

data, those plans and test data can probably be used in the testing effort, thereby reducing the time and cost of testing.

1.9.1.2 Functional and Structural Testing

While testing your project team's solution, your testers will perform functional or structural tests with the verification and validation techniques just described. Functional testing is sometimes called black-box testing because no knowledge of the internal logic of the system is used to develop test cases. For example, if a certain function key should produce a specific result when pressed, a functional test validates this expectation by pressing the function key and observing the result. When conducting functional tests, you'll use validation techniques almost exclusively.

Conversely, structural testing is sometimes called white-box testing because knowledge of the internal logic of the system is used to develop hypothetical test cases. Structural tests use verification predominantly. If a software development team creates a block of code that will allow a system to process information in a certain way, a test team would verify this structurally by reading the code, and given the system's structure, see if the code would work reasonably. If they felt it could, they would plug the code into the system and run an application to structurally validate the code. Each method has its pros and cons:

- Functional Testing Advantages
  - Simulates actual system usage.
  - Makes no system structure assumptions
- Functional Testing Disadvantages
  - Potential of missing logical errors in software
  - Possibility of redundant testing
- Structural Testing Advantages
  - You can test the software's structure logic
  - You test code that you wouldn't use if you performed only functional testing
- Structural Testing Disadvantages
  - Does not ensure that you've met user requirements
  - Its tests may not mimic real-world situations

1.9.1.2.1 Why Use Both Testing Methods?

Both methods together validate the entire system as shown in Table 1-9. For example, a functional test case might be taken from the documentation description of how to perform a certain function, such as accepting bar code input. A structural test case might be taken from a technical documentation manual. To effectively test systems, you need to use both methods.

Table 1-9: Functional Testing

Test Phase	Performed by:	Verification	Validation
Feasibility Review	Developers, Users	X	
Requirements Review	Developers, Users	X	
Unit Testing	Developers		X
Integrated Testing	Developers		X
System Testing	Developers with User Assistance		X

1.9.2 Static versus Dynamic Testing

Static testing is performed using the software documentation. The code is not executing during static testing. Dynamic testing requires the code to be in an executable state to perform the tests.

Most verification techniques are static tests.

- Feasibility Reviews – Tests for this structural element would verify the logic flow of a unit of software.
- Requirements Reviews – These reviews verify software relationships; for example, in any particular system, the structural limits of how much load (e.g., transactions or number of concurrent users) a system can handle.

Most validation tests are dynamic tests. Examples of this are:

- Unit Testing

  These tests verify that the system functions properly; for example, pressing a function key to complete an action.

- Integrated Testing

  The system runs tasks that involve more than one application or database to verify that it performed the tasks accurately.

- System Testing

  The tests simulate operation of the entire system and verify that it ran correctly.

- User Acceptance

  This real-world test means the most to your business, and unfortunately, there's no way to conduct it in isolation. Once your organization's staff, customers, or vendors begin to interact with your system, they'll verify that it functions properly for you.

1.9.3 The "V" Concept of Testing

Life cycle testing involves continuous testing of the system during the developmental process. At predetermined points, the results of the development process are inspected to determine the correctness of the implementation. These inspections identify defects at the earliest possible point.

Life cycle testing cannot occur until a formalized Systems Development Life Cycle (SDLC) has been incorporated. Life cycle testing is dependent upon the completion of predetermined deliverables at specified points in the developmental life cycle. If information services personnel have the discretion to determine the order in which deliverables are developed, the life cycle test process becomes ineffective. This is due to variability in the process, which normally increases cost.

The life cycle testing concept can best be accomplished by the formation of a test team. The team is comprised of members of the project who may be both implementing and testing the system. When members of the team are testing the system, they must use a formal testing methodology to clearly distinguish the implementation mode from the test mode. They also must follow a structured methodology when approaching testing the same as when approaching system development. Without a specific structured test methodology, the test team concept is ineffective because team members would follow the same methodology for testing as they used for developing the system. Experience shows people are blind to their own mistakes, so the effectiveness of the test team is dependent upon developing the system under one methodology and testing it under another.

The life cycle testing concept is illustrated in Figure 1-20. This illustration shows that when the project starts both the system development process and system test process begins. The team that is developing the system begins the systems development process and the team that is conducting the system test begins planning the system test process. Both teams start at the same point using the same information. The systems development team has the responsibility to define and document the requirements for developmental purposes. The test team will likewise use those same requirements, but for the purpose of testing the system. At appropriate points during the developmental process, the test team will test the developmental process in an attempt to uncover defects. The test team should use the structured testing techniques outlined in this book as a basis of evaluating the system development process deliverables.

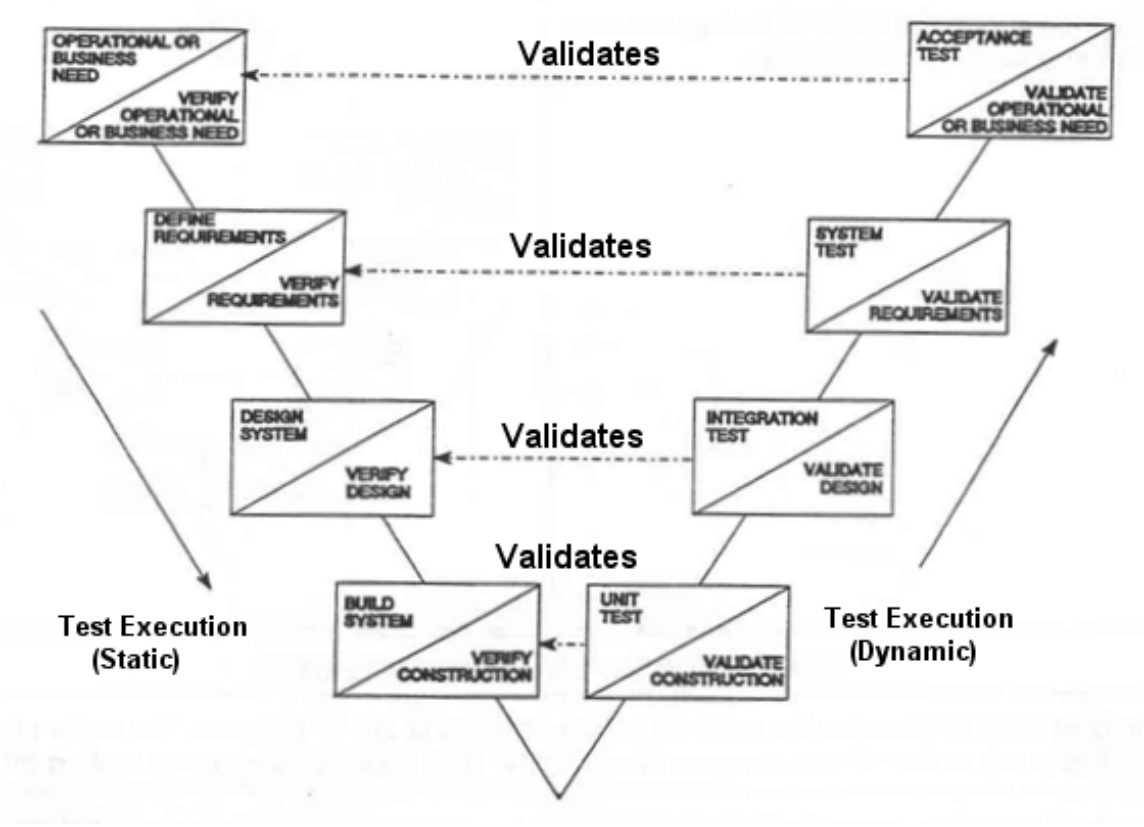


Figure 1-20 The "V" Concept of Software Testing

During the system test process, an appropriate set of test transactions should be developed to be completed at the same time as the completion of the application system. When the application meets the acceptance criteria, it can be integrated into the operating environment. During this process, the systems development team and the systems test team work closely together to ensure that the application is properly integrated into the production environment. At that point, the teams again split to ensure the correctness of changes made during the maintenance phase. The maintenance team will make whatever changes and enhancements are necessary to the application system, and the test team will continue the test process to ensure that those enhancements are properly implemented and integrated into the production environment.

In the V-testing concept, your project's Do and Check procedures slowly converge from start to finish (see Figure 1-21 The 11-Step Software Testing Process Example), which indicates that as the Do team attempts to implement a solution, the Check team concurrently develops a process to minimize or eliminate the risk. If the two groups work closely together, the high level of risk at a project's inception will decrease to an acceptable level by the project's conclusion.

1.9.3.1 An 11-Step Software Testing Process Example

The software testing process, as illustrated in Figure 1-21, is an 11-step testing process that follows the “V” concept of testing. The “V” represents both the software development process and the 11-step software testing process. The first five steps use verification as the primary means to evaluate the correctness of the interim development deliverables. Validation is used to test the software in an executable mode. Results of both verification and validation should be documented. Both verification and validation will be used to test the installation of the software as well as changes to the software. The final step of the “V” process represents both the development and test team evaluating the effectiveness of testing.

Note: The terms in this example vary slightly from the SDLC example to illustrate there are no common definitions used by all IT organizations.

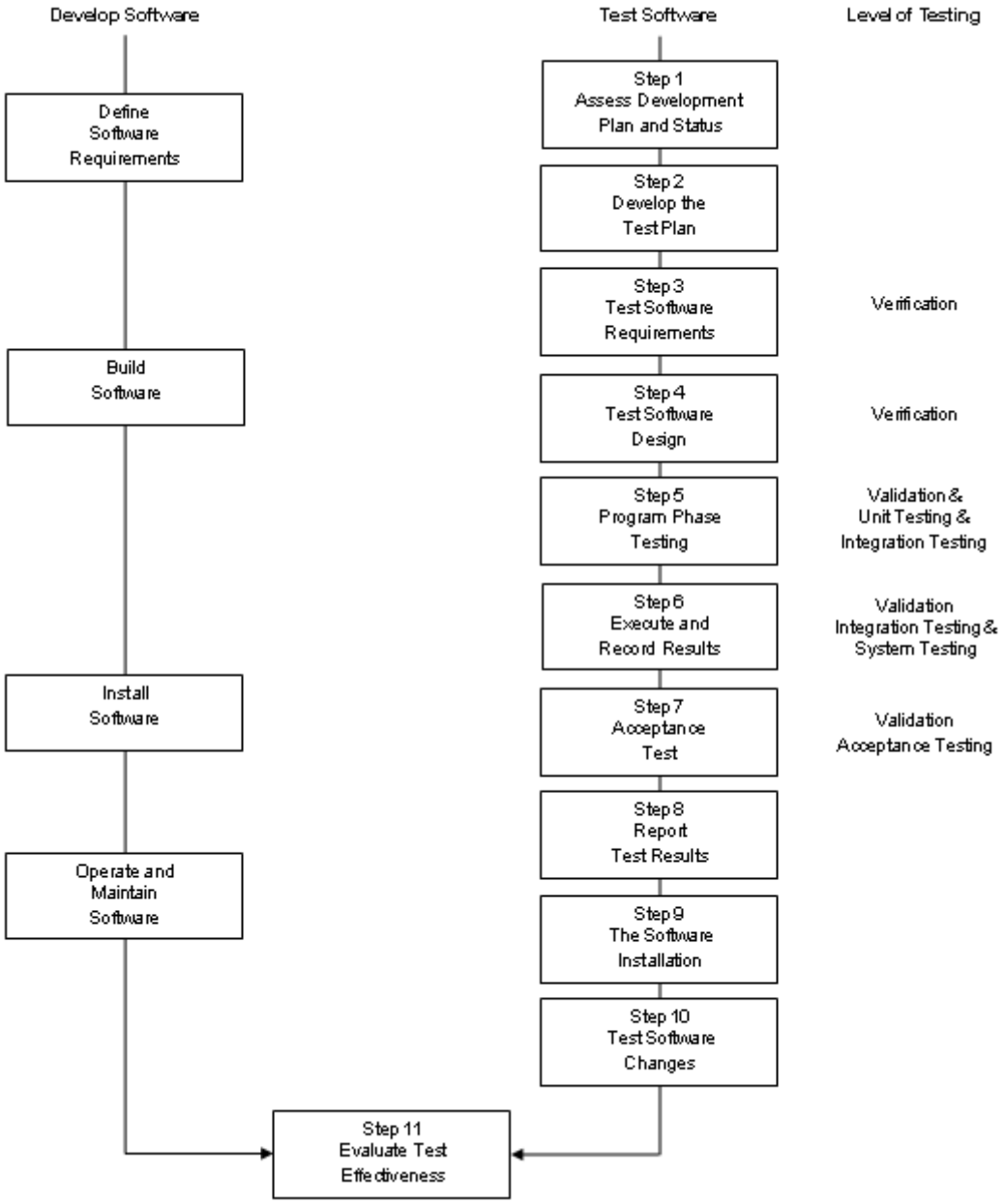


Figure 1-21 The 11-Step Software Testing Process Example

1.9.3.1.1 Step 1: Assess Development Plan and Status

This first step is a prerequisite to building the Verification, Validation, and Testing (VV&T) Plan used to evaluate the implemented software solution. During this step, testers challenge the completeness and correctness of the development plan. Based on the extensiveness and completeness of the Project Plan the testers can estimate the amount of resources they will need to test the implemented software solution.

1.9.3.1.2 Step 2: Develop the Test Plan

Forming the plan for testing will follow the same pattern as any software planning process. The structure of all plans should be the same, but the content will vary based on the degree of risk the testers perceive as associated with the software being developed.

1.9.3.1.3 Step 3: Test Software Requirements

Incomplete, inaccurate, or inconsistent requirements lead to most software failures. The inability to get requirements right during the requirements gathering phase can also increase the cost of implementation significantly. Testers, through verification, must determine that the requirements are accurate, complete, and they do not conflict with one another.

1.9.3.1.4 Step 4: Test Software Design

This step tests both external and internal design primarily through verification techniques. The testers are concerned that the design will achieve the objectives of the requirements, as well as the design being effective and efficient on the designated hardware.

1.9.3.1.5 Step 5: Program (Build) Phase Testing

The method chosen to build the software from the internal design document will determine the type and extensiveness of tests needed. As the construction becomes more automated, less testing will be required during this phase. However, if software is constructed using the waterfall process, it is subject to error and should be verified. Experience has shown that it is significantly cheaper to identify defects during the construction phase, than through dynamic testing during the test execution step.

1.9.3.1.6 Step 6: Execute and Record Results

This involves the testing of code in a dynamic state. The approach, methods, and tools specified in the test plan will be used to validate that the executable code in fact meets the stated software requirements, and the structural specifications of the design.

1.9.3.1.7 Step 7: Acceptance Test

Acceptance testing enables users to evaluate the applicability and usability of the software in performing their day-to-day job functions. This tests what the user believes the software should perform, as opposed to what the documented requirements state the software should perform.

1.9.3.1.8 Step 8: Report Test Results

Test reporting is a continuous process. It may be both oral and written. It is important that defects and concerns be reported to the appropriate parties as early as possible, so that corrections can be made at the lowest possible cost.

1.9.3.1.9 Step 9: The Software Installation

Once the test team has confirmed that the software is ready for production use, the ability to execute that software in a production environment should be tested. This tests the interface to operating software, related software, and operating procedures.

1.9.3.1.10 Step 10: Test Software Changes

While this is shown as Step 10, in the context of performing maintenance after the software is implemented, the concept is also applicable to changes throughout the implementation process. Whenever requirements change, the test plan must change, and the impact of that change on software systems must be tested and evaluated.

1.9.3.1.11 Step 11: Evaluate Test Effectiveness

Testing improvement can best be achieved by evaluating the effectiveness of testing at the end of each software test assignment. While this assessment is primarily performed by the testers, it should involve the developers, users of the software, and quality assurance professionals if the function exists in the IT organization.

1.10 Testing Techniques

Testing techniques are the means used by testers to accomplish their test objectives. This section addresses the following techniques:

- Structural versus Functional Technique Categories
- Examples of Specific Testing Techniques

1.10.1 Structural versus Functional Technique Categories

The properties that the test set is to reflect are classified according to whether they are derived from a description of the program's function or from the program's internal structure. Both structural and functional analysis should be performed to ensure adequate testing. Structural analysis-based test sets tend to uncover errors that occur during "coding" of the program, while functional analysis-based test sets tend to uncover errors that occur in implementing requirements or design specifications.

Functional testing ensures that the requirements are properly satisfied by the application system. The functions are those tasks that the system is designed to accomplish. Functional testing is not concerned with how processing occurs, but rather, with the results of processing.

Structural testing ensures sufficient testing of the implementation of a function. Although used primarily during the coding phase, structural analysis should be used in all phases of the life cycle where the software is represented formally in some algorithmic, design, or

requirements language. The intent of structural testing is to assess the implementation by finding test data that will force sufficient coverage of the structures present in the implemented application. Structural testing evaluates that all aspects of the structure have been tested and that the structure is sound. Determining that all tasks through a structure are tested is a difficult process and one requiring extensive test data. However, determining if the structure functions properly is a test task that is more easily accomplished.

1.10.1.1 Structural System Testing Technique Categories

Structural system testing is designed to verify that the developed system and programs work. The objective is to ensure that the product designed is structurally sound and will function correctly. It attempts to determine that the technology has been used properly and that when all the component parts are assembled they function as a cohesive unit. The structural system testing techniques provide the facility for determining that the implemented configuration and its interrelationship of parts functions so that they can perform the intended tasks. The techniques are not designed to ensure that the application system is functionally correct, but rather, that it is structurally sound. The structural system testing techniques are briefly described in Table 1-10.

Table 1-10: Structural Testing Techniques

Technique	Description	Example
Stress	Determine system performs with expected volumes.	• Sufficient disk space allocated • Communication lines adequate
Execution	System achieves desired level of proficiency.	• Transaction turnaround time adequate • Software/hardware use optimized
Recovery	System can be returned to an operational status after a failure.	• Induce failure • Evaluate adequacy of backup data
Operations	System can be executed in a normal operational status.	• Determine systems can run using document • JCL adequate
Compliance (to Process)	System is developed in accordance with standards and procedures.	• Standards followed • Documentation complete
Security	System is protected in accordance with importance to organization.	• Access denied • Procedures in place

1.10.1.1.1 Stress Testing Techniques

Stress testing is designed to determine if the system can function when subject to large volumes of work i.e., larger than would be normally expected. The areas that are stressed include input transactions, internal tables, disk space, output, communications, computer

capacity, and interaction with people. If the application functions adequately under test, it can be assumed that it will function properly with normal volumes of work.

Objectives

The objective of stress testing is to simulate a production environment for the purpose of determining that:

- Normal or above-normal volumes of transactions can be processed through the transaction within the expected time frame.
- The application system is structurally able to process large volumes of data.
- System capacity, including communication lines, has sufficient resources available to meet expected turnaround times.
- People can perform their assigned tasks and maintain the desired turnaround time.

How to Use Stress Testing

Stress testing should simulate as closely as possible the production environment. Online systems should be stress tested by having people enter transactions at a normal or above normal pace. Batch systems can be stress tested with large input batches. Error conditions should be included in tested transactions. Transactions for use in stress testing can be obtained from one of the following three sources:

- Test data generators
- Test transactions created by the test group
- Transactions previously processed in the production environment

In stress testing, the system should be run as it would in the production environment. Operators should use standard documentation, and the people entering transactions or working with the system should be the clerical personnel that will work with the system after it goes into production. Online systems should be tested for an extended period of time, and batch systems tested using more than one batch of transactions.

Stress Test Example

Stress tests can be designed to test all or parts of an application system. For example, stress testing might:

- Enter transactions to determine that sufficient disk space has been allocated to the application.
- Ensure that the communication capacity is sufficient to handle the volume of work by attempting to overload the network with transactions.
- Test system overflow conditions by entering more transactions than can be accommodated by tables, queues, and internal storage facilities, etc.

When to Use Stress Testing

Stress testing should be used when there is uncertainty regarding the amount of work the application system can handle without failing. Stress testing attempts to break the system by

overloading it with a large volume of transactions. Stress testing is most common with online applications because it is difficult to simulate heavy volume transactions using the other testing techniques. The disadvantage of stress testing is the amount of time it takes to prepare for the test, plus the amount of resources consumed during the actual execution of the test. These costs need to be weighed against the risk of not identifying volume-related failures until the application is placed into an operational mode.

1.10.1.1.2 Execution Testing Technique

Execution testing determines whether the system achieves the desired level of proficiency in a production status. Execution testing can verify response times, turnaround times, as well as design performance. The execution of a system can be tested in whole or in part, using the actual system or a simulated model of a system.

Objectives

Execution testing is used to determine whether the system can meet the specific performance criteria. The objectives of execution testing include:

- Determine the performance of the system structure.
- Verify the optimum use of hardware and software.
- Determine the response time to online user requests.
- Determine transaction processing turnaround time.

How to Use Execution Testing

Execution testing can be conducted in any phase of the system development life cycle. The testing can evaluate a single aspect of the system, for example, a critical routine in the system, or the ability of the proposed structure to satisfy performance criteria. Execution testing can be performed in any of the following manners:

- Using hardware and software monitors
- Simulating the functioning of all or part of the system using a simulation model
- Creating a quick and dirty program(s) to evaluate the approximate performance of a completed system

Execution testing may be executed on-site or off-site for the performance of the test. For example, execution testing can be performed on hardware and software before being acquired, or may be done after the application system has been completed. The earlier the technique is used, the higher the assurance that the completed application will meet the performance criteria.

Execution Test Examples

Examples of the use of execution testing include:

- Calculating turnaround time on transactions processed through the application
- Determining that the hardware and software selected provide the optimum processing capability

- Using software monitors to determine that the program code is effectively used

When to Use Execution Testing

Execution testing should be used early in the developmental process. While there is value in knowing that the completed application does not meet performance criteria, if that assessment is not known until the system is operational, it may be too late or too costly to make the necessary modifications. Therefore, execution testing should be used at that point in time when the results can be used to affect or change the system structure.

1.10.1.1.3 Recovery Testing Technique

Recovery is the ability to restart operations after the integrity of the application has been lost. The process normally involves reverting to a point where the integrity of the system is known, and then reprocessing transactions up until the point of failure. The time required to recover operations is affected by the number of restart points, the volume of applications run on the computer center, the training and skill of the people conducting the recovery operation, and the tools available for recovery. The importance of recovery will vary from application to application.

Objectives

Recovery testing is used to ensure that operations can be continued after a disaster. Recovery testing not only verifies the recovery process, but also the effectiveness of the component parts of that process. Specific objectives of recovery testing include:

- Preserve adequate backup data
- Store backup data in a secure location
- Document recovery procedures
- Assign and train recovery personnel
- Develop recovery tools and make available

How to Use Recovery Testing

Recovery testing can be conducted in two modes. First, the procedures, methods, tools, and techniques can be assessed to evaluate whether they appear adequate; and second, after the system has been developed, a failure can be introduced into the system and the ability to recover tested. Both types of recovery testing are important. The implementation of the technique is different depending upon which type of recovery testing is being performed.

Evaluating the procedures and documentation is a process using primarily judgment and checklists. On the other hand, the actual recovery test may involve off-site facilities and alternate processing locations. Testing the procedures is normally done by skilled systems analysts, professional testers, or management personnel. On the other hand, testing the actual recovery procedures should be performed by computer operators and other clerical personnel, who would be involved had there been an actual disaster instead of a test disaster.

A simulated disaster is usually performed on one aspect of the application system. For example, the test may be designed to determine whether people using the system can continue processing and recover computer operations after computer operations cease. While several aspects of recovery need to be tested, it is better to test one segment at a time rather than induce multiple failures at a single time. When multiple failures are induced, and problems are encountered, it may be more difficult to pinpoint the cause of the problem than when only a single failure is induced.

It is preferable not to advise system participants when a disaster test will be conducted. For example, a failure might be intentionally introduced during a normal system test to observe reaction and evaluate the recovery test procedures. When people are prepared, they may perform the recovery test in a manner different from the performance when it occurs at an unexpected time. Even if the participants know that recovery may be part of the test, it is not recommended to let them know specifically when it will occur, or what type of recovery will be necessary.

Recovery Test Example

Recovery testing can involve the manual functions of an application, loss of input capability, loss of communication lines, hardware or operating system failure, loss of database integrity, operator error, or application system failure. It is desirable to test all aspects of recovery processing. One example of recovery testing would entail inducing a failure into one of the application system programs during processing. This could be accomplished by inserting a special instruction to look for a transaction code that upon identification would cause an abnormal program termination.

The recovery could be conducted from a known point of integrity to ensure that the available backup data was adequate for the recovery process. When the recovery had been completed, the files at the point where the exercise was requested could be compared to the files recreated during the recovery process.

When to Use Recovery Testing

Recovery testing should be performed whenever the user of the application states that the continuity of operation of the application is essential to the proper functioning of the user area. The user should estimate the potential loss associated with inability to recover operations over various time spans; for example, the inability to recover within five minutes, one hour, eight hours, and a week. The amount of the potential loss should both determine the amount of resource to be put into disaster planning as well as recovery testing.

1.10.1.1.4 Operations Testing Technique

After testing, the application will be integrated into the operating environment. At this point in time, the application will be executed using the normal operation staff, operations procedures, and documentation. Operations' testing is designed to verify prior to production that the operating procedures and staff can properly execute the application.

Objectives

Operations' testing is primarily designed to determine whether the system is executable during normal systems operations. The specific objectives include:

- Determine the completeness of computer operator documentation
- Ensure that the necessary support mechanisms, such as job control language, are prepared and function properly
- Evaluate the completeness of operator training
- Test to ensure that operators using prepared documentation can, in fact, operate the system

How to Use Operations Testing

Operations' testing evaluates both the process and the execution of the process. During the requirements phase, operational requirements can be evaluated to determine the reasonableness and completeness of those requirements. During the design phase, the operating procedures should be designed and thus can be evaluated. This continual definition of the operating procedures should be subjected to continual testing.

The execution of operations testing can normally be performed in conjunction with other tests. However, if operations' testing is included, the operators should not be prompted or helped by outside parties during the test process. The test needs to be executed as if it was part of normal computer operations in order to adequately evaluate the effectiveness of computer operators in running the application in a true-to-life operations environment.

Operations Testing Example

Operations' testing is a specialized technical test of executing the application system and includes:

- Determining that the operator instructions have been prepared and documented in accordance with other operations instructions, and that computer operators have been trained in any unusual procedures
- Testing that the job control language statements and other operating systems support features perform the predetermined tasks
- Verifying that the file labeling and protection procedures function properly

When to Use Operations Testing

Operations' testing should occur prior to placing any application into a production status. If the application is to be tested in a production-type setting, operations testing can piggyback that process at a very minimal cost. It is as important to identify an operations flaw as it is an application flaw prior to placing the application into production.

1.10.1.1.5 Compliance Testing Technique

Compliance testing verifies that the application was developed in accordance with information technology standards, procedures, and guidelines. The methodologies are used to

increase the probability of success, to enable the transfer of people in and out of the project with minimal cost, and to increase the maintainability of the application system. The type of testing conducted varies on the phase of the systems development life cycle. However, it may be more important to compliance test adherence to the process during requirements than at later stages in the life cycle because it is difficult to correct applications when requirements are not adequately documented.

Objectives

Compliance testing is performed to both ensure compliance to the methodology and to encourage and help the information technology professional comply with the methodology. Specific compliance objectives include:

- Determine that systems development and maintenance methodologies are followed
- Ensure compliance to departmental standards, procedures, and guidelines
- Evaluate the completeness and reasonableness of application system documentation

How to Use Compliance Testing

Compliance testing requires that the prepared document or program is compared to the standards for that particular program or document. A colleague would be the most appropriate person to do this comparison. The most effective method of compliance testing is the inspection process.

Compliance Testing Examples

A peer group of programmers would be assembled to test line-by-line that a computer program is compliant with programming standards. At the end of the peer review, the programmer would be given a list of noncompliant information that would need to be corrected.

When to Use Compliance Testing

Compliance to information technology application system development standards and procedures is dependent upon management's desire to have the procedures followed and the standards enforced. Therefore, if management really wants compliance they should perform sufficient tests to determine both the degree of compliance with the methodology and to identify violators for management action. However, lack of compliance should also be used from the perspective that the standards may be misunderstood, not adequately instructed or publicized, or may, in fact, be poor standards inhibiting the development of application systems. In these instances, it may be desirable to change the methodology.

1.10.1.1.6 Security Testing Technique

Security is a protection system that is needed for both secure confidential information and for competitive purposes to assure third parties their data will be protected. The amount of security provided will be dependent upon the risks associated with compromise or loss of information. Protecting the confidentiality of the information is designed to protect the resources of the organization. However, information such as customer lists or improper

disclosure of customer information may result in a loss of customer business to competitors. Security testing is designed to evaluate the adequacy of the protective procedures and countermeasures.

Objectives

Security defects do not become as obvious as other types of defects. Therefore, the objectives of security testing are to identify defects that are very difficult to identify. Even failures in the security system operation may not be detected, resulting in a loss or compromise of information without the knowledge of that loss. The security testing objectives include:

- Determine that adequate attention is devoted to identifying security risks
- Determine that a realistic definition and enforcement of access to the system is implemented
- Determine that sufficient expertise exists to perform adequate security testing
- Conduct reasonable tests to ensure that the implemented security measures function properly

How to Use Security Testing Techniques

Security testing is a highly specialized part of the test process. Most organizations can evaluate the reasonableness of security procedures to prevent the average perpetrator from penetrating the application. However, the highly skilled perpetrator using sophisticated techniques may use methods undetectable by novices designing security measures and/or testing those measures.

The first step in testing is the identification of the security risks and the potential loss associated with those risks. If either the loss is low or the penetration method mere routine, the information technology personnel can conduct the necessary tests. On the other hand, if either the risks are very high or the technology that might be used is sophisticated, specialized help should be acquired in conducting the security tests.

Security Test Example

Security testing involves a wide spectrum of conditions. Testing can first be divided into physical and logical security. Physical deals with the penetration by people in order to physically gather information, while logical security deals with the use of computer processing and/or communication capabilities to improperly access information. Second, access control can be divided by type of perpetrator, such as employee, consultant, cleaning or service personnel, as well as categories of employees. The type of test conducted will vary upon the condition being tested and can include:

- Determination that the resources being protected are identified, and access is defined for each resource. Program or individual can define access.
- Evaluation as to whether the designed security procedures have been properly implemented and function in accordance with the specifications.
- Unauthorized access can be attempted in online systems to ensure that the system can identify and prevent access by unauthorized sources.

When to Use Security Testing

Security testing should be used when the information and/or assets protected by the application system are of significant value to the organization. The testing should be performed both prior to the system going into an operational status and after the system is placed into an operational status. The extent of testing should depend on the security risks, and the individual assigned to conduct the test should be selected based on the estimated sophistication that might be used to penetrate security.

1.10.1.2 Functional System Testing Technique Categories

Functional system testing ensures that the system requirements and specifications are achieved. The process normally involves creating test conditions for use in evaluating the correctness of the application. The types of techniques useful in performing functional testing techniques are briefly described in Table 1-11.

Table 1-11: Functional Testing Techniques

Technique	Description	Example
Requirements	System performs as specified.	Prove system requirements. Compliance to policies, regulations.
Regression	Verifies that anything unchanged still performs correctly.	Unchanged system segments function. Unchanged manual procedures correct.
Error Handling	Errors can be prevented or detected, and then corrected.	Error introduced into test. Errors re-entered.
Manual Support	The people-computer interaction works.	Manual procedures developed. People trained.
Intersystem	Data is correctly passed from system to system.	Intersystem parameters changed. Intersystem documentation updated.
Control	Controls reduce system risk to an acceptable level.	File reconciliation procedures work. Manual controls in place.
Parallel	Old system and new system are run and the results compared to detect unplanned differences.	Old and new systems can reconcile. Operational status of old system maintained.

1.10.1.2.1 Requirements Testing Techniques

Requirements testing must verify that the system can perform its function correctly and that the correctness can be sustained over a continuous period of time. Unless the system can function correctly over an extended period of time, management will not be able to rely upon the system. The system can be tested for correctness throughout the life cycle, but it is difficult to test the reliability until the program becomes operational.

Objectives

Successfully implementing user requirements is only one aspect of requirements testing. The responsible user is normally only one of many groups having an interest in the application system. The objectives that need to be addressed in requirements testing are:

- Implement user requirements
- Maintain correctness over extended processing periods
- Ensure that application processing complies with the organization's policies and procedures

Secondary user needs have been included, such as:

- Security officer
- Database administrator
- Internal auditors
- Records retention
- Comptroller
- System processes accounting information in accordance with generally accepted accounting procedures.
- Application systems process information in accordance with governmental regulations.

How to Use Requirements Testing

Requirements' testing is primarily performed through the creation of test conditions and functional checklists. Test conditions are generalized during requirements, and become more specific as the SDLC progresses, leading to the creation of test data for use in evaluating the implemented application system.

Functional testing is more effective when the test conditions are created directly from user requirements. When test conditions are created from the system documentation, defects in that documentation will not be detected through testing. When the test conditions are created from other than the system documentation, defects introduced into the documentation will be detected.

Requirements Test Example

Typical requirement test examples include:

- Creating a test matrix to prove that the systems requirements as documented are the requirements desired by the user
- Using a checklist prepared specifically for the application to verify the application's compliance to organizational policies and governmental regulations
- Determining that the system meets the requirements established by the organization's department of internal auditors

When to Use Requirements Testing

Every application should be requirements tested. The process should begin in the requirements phase, and continue through every phase of the life cycle into operations and maintenance. It is not a question as to whether requirements must be tested but, rather, the extent and methods used in requirements testing.

1.10.1.2.2 Regression Testing Technique

One of the attributes that has plagued information technology professionals for years is the snowballing or cascading effect of making changes to an application system. One segment of the system is developed and thoroughly tested. Then a change is made to another part of the system, which has a disastrous effect on the thoroughly tested portion. Either the incorrectly implemented change causes a problem, or the change introduces new data or parameters that cause problems in a previously tested segment. Regression testing retests previously tested segments to ensure that they still function properly after a change has been made to another part of the application.

Objectives

Regression testing involves assurance that all aspects of an application system remain functional after testing. The introduction of change is the cause of problems in previously tested segments. The objectives of regression testing include:

- Determine whether systems documentation remains current
- Determine that system test data and test conditions remain current
- Determine that previously tested system functions perform properly after changes are introduced into the application system

How to Use Regression Testing

Regression testing is retesting unchanged segments of the application system. It normally involves rerunning tests that have been previously executed to ensure that the same results can be achieved currently as were achieved when the segment was last tested. While the process is simple in that the test transactions have been prepared and the results known, unless the process is automated it can be a very time-consuming and tedious operation. It is also one in which the cost/benefit needs to be carefully evaluated or large amounts of effort can be expended with minimal payback.

Regression Test Example

Examples of regression testing include:

- Rerunning of previously conducted tests to ensure that the unchanged system segments function properly
- Reviewing previously prepared manual procedures to ensure that they remain correct after changes have been made to the application system
- Obtaining a printout from the data dictionary to ensure that the documentation for data elements that have been changed is correct

When to Use Regression Testing

Regression testing should be used when there is a high risk that new changes may affect unchanged areas of the application system. In the developmental process, regression testing should occur after a predetermined number of changes are incorporated into the application system. In maintenance, regression testing should be conducted if the potential loss that could occur due to affecting an unchanged portion is very high. The determination as to whether to conduct regression testing should be based upon the significance of the loss that could occur due to improperly tested applications.

1.10.1.2.3 Error-Handling Testing Technique

One characteristic that differentiates automated from manual systems is the predetermined error-handling features. Manual systems can deal with problems as they occur, but automated systems must preprogram error-handling. In many instances the completeness of error-handling affects the usability of the application. Error-handling testing determines the ability of the application system to properly process incorrect transactions.

Objectives

Errors encompass all unexpected conditions. In some systems, approximately 50 percent of the programming effort will be devoted to handling error conditions. Specific objectives of error-handling testing include:

- Determine that all reasonably expected error conditions are recognizable by the application system
- Determine that the accountability for processing errors has been assigned and that the procedures provide a high probability that the error will be properly corrected
- Determine that reasonable control is maintained over errors during the correction process

How to Use Error-Handling Testing

Error-handling testing requires a group of knowledgeable people to anticipate what can go wrong with the application system. Most other forms of testing involve verifying that the application system conforms to requirements. Error-handling testing uses exactly the opposite concept.

A successful method for developing test error conditions is to assemble, for a half-day or a day, people knowledgeable in information technology, the user area, and auditing or error-tracking. These individuals are asked to brainstorm what might go wrong with the application. The totality of their thinking must then be organized by application function so that a logical set of test transactions can be created. Without this type of synergistic interaction on errors, it is difficult to develop a realistic body of problems prior to production.

Error-handling testing should test the introduction of the error, the processing of the error, the control condition, and the reentry of the condition properly corrected. This requires error-handling testing to be an iterative process in which errors are first introduced into the system,

then corrected, then reentered into another iteration of the system to satisfy the complete error-handling cycle.

Error-Handling Test Examples

Error-handling requires you to think negatively, and conduct such tests as:

- Produce a representative set of transactions containing errors and enter them into the system to determine whether the application can identify the problems.
- Through iterative testing, enter errors that will result in corrections, and then reenter those transactions with errors that were not included in the original set of test transactions.
- Enter improper master data, such as prices or employee pay rates, to determine if errors that will occur repetitively are subjected to greater scrutiny than those causing single error results.

When to Use Error-Handling Testing

Error testing should occur throughout the system development life cycle. At all points in the developmental process the impact from errors should be identified and appropriate action taken to reduce those errors to an acceptable level. Error-handling testing assists in the error management process of systems development and maintenance. Some organizations use auditors, quality assurance, or professional testing personnel to evaluate error processing.

1.10.1.2.4 Manual Support Testing Techniques

Systems commence when transactions originate and conclude with the use of the results of processing. The manual part of the system requires the same attention to testing, as does the automated segment. Although the timing and testing methods may be different, the objectives of manual testing remain the same as testing the automated segment of the application system.

Objectives

Manual support involves all the functions performed by people in preparing data for, and using data from, automated applications. The objectives of testing the manual support systems are:

- Verify that the manual support procedures are documented and complete
- Determine that manual support responsibility has been assigned.
- Determine that the manual support people are adequately trained
- Determine that the manual support and the automated segment are properly interfaced

How to Use Manual Support Testing

Manual testing involves the evaluation of the adequacy of the process first, and then, second, the execution of the process. The process itself can be evaluated in all segments of the systems development life cycle. The execution of the process can be done in conjunction with normal systems testing. Rather than prepare and enter test transactions, the system can be tested

having the actual clerical and supervisory people prepare, enter, and use the results of processing from the application system.

Manual testing normally involves several iterations of the process. To test people processing requires testing the interface between people and the application system. This means entering transactions, and then getting the results back from that processing, making additional actions based on the information received, until all aspects of the manual computer interface have been adequately tested.

The manual support testing should occur without the assistance of the systems personnel. The manual support group should operate using the training and procedures provided them by the systems personnel. However, the systems personnel should evaluate the results to determine if the tests have been adequately performed.

Manual Support Test Example

The process of conducting manual support testing can include the following types of tests:

- Provide input personnel with the type of information they would normally receive from their customers and then have them transcribe that information and enter it into the computer.
- Output reports are prepared from the computer based on typical conditions, and the users are then asked to take the necessary action based on the information contained in computer reports.
- Users can be provided a series of test conditions and then asked to respond to those conditions. Conducted in this manner, manual support testing is like an examination in which the users are asked to obtain the answer from the procedures and manuals available to them.

When to Use Manual Support Testing

Verification that the manual systems function properly should be conducted throughout the systems development life cycle. This aspect of system testing should not be left to the latter stages of the life cycle to test. However, extensive manual support testing is best done during the installation phase so that the clerical people do not become involved with the new system until immediately prior to its entry into operation. This avoids the confusion of knowing two systems and not being certain which rules to follow. During the maintenance and operation phases, manual support testing may only involve providing people with instructions on the changes and then verifying with them through questioning that they properly understand the new procedures.

1.10.1.2.5 Intersystem Testing Technique

Application systems are frequently interconnected to other application systems. The interconnection may be data coming into the system from another application, leaving for another application, or both. Frequently multiple applications – sometimes called cycles or functions – are involved. For example, there is a revenue function or cycle that interconnects all of the income-producing applications such as order entry, billing, receivables, shipping,

and returned goods. Intersystem testing is designed to ensure that the interconnection between applications functions correctly.

Objectives

Many problems exist in intersystem testing. One is that it is difficult to find a single individual having jurisdiction over all of the systems below the level of senior management. In addition, the process is time-consuming and costly. The objectives of intersystem testing include:

- Determine that proper parameters and data are correctly passed between applications
- Ensure that proper coordination and timing of functions exists between the application systems
- Determine that documentation for the involved systems is accurate and complete

How to Use Intersystem Testing

Intersystem testing involves the operation of multiple systems in the test. Thus, the cost may be expensive, especially if the systems have to be run through several iterations. The process is not difficult; in that files or data used by multiple systems are passed from one another to verify that they are acceptable and can be processed properly. However, the problem can be magnified during maintenance when two or more of the systems are undergoing internal changes concurrently.

One of the best testing tools for intersystem testing is the integrated test facility. This permits testing to occur during a production environment and thus the coupling of systems can be tested at minimal cost. The integrated test facility is described in Skill Category 2, Building the Test Environment.

Intersystem Test Example

Procedures used to conduct intersystem testing include:

- Developing a representative set of test transactions in one application for passage to another application for processing verification.
- Entering test transactions in a live production environment using the integrated test facility so that the test conditions can be passed from application to application to application, to verify that the processing is correct.
- Manually verifying that the documentation in the affected systems is updated based upon the new or changed parameters in the system being tested.

When to Use Intersystem Testing

Intersystem testing should be conducted whenever there is a change in parameters between application systems. The extent and type of testing will depend on the risk associated with those parameters being erroneous. If the integrated test facility concept is used, the intersystem parameters can be verified after the changed or new application is placed into production.

1.10.1.2.6 Control Testing Technique

Approximately one-half of the total system development effort is directly attributable to controls. Controls include data validation, file integrity, audit trail, backup and recovery, documentation, and the other aspects of systems related to integrity. Control testing techniques are designed to ensure that the mechanisms that oversee the proper functioning of an application system work.

Objectives

Control is a management tool to ensure that processing is performed in accordance with the intents of management. The objectives of control include:

- Data is accurate and complete
- Transactions are authorized
- An adequate audit trail of information is maintained.
- The process is efficient, effective, and economical
- The process meets the needs of the user

How to Use Control Testing

Control can be considered a system within a system. The term "system of internal controls" is frequently used in accounting literature to describe the totality of the mechanisms that ensure the integrity of processing. Controls are designed to reduce risks; therefore, in order to test controls the risks must be identified. The individual designing the test then creates the risk situations in order to determine whether the controls are effective in reducing them to a predetermined acceptable level of risk.

One method that can be used in testing controls is to develop a risk matrix. The matrix identifies the risks, the controls, and the segment within the application system in which the controls reside. The risk matrix is described in Skill Category 4, Risk Analysis.

Control Testing Example

Control-oriented people frequently do control testing. Like error-handling, it requires a negative look at the application system to ensure that those "what-can-go-wrong" conditions are adequately protected. Error-handling is a subset of controls oriented toward the detection and correction of erroneous information. Control in the broader sense looks at the totality of the system.

Examples of testing that might be performed to verify controls include:

- Determine that there is adequate assurance that the detailed records in a file equal the control total. This is normally done by running a special program that accumulates the detail and reconciles it to the total.
- Determine that the manual controls used to ensure that computer processing is correct are in place and working.

On a test basis, select transactions and verify that the processing for those transactions can be reconstructed.

When to Use Control Testing

Control testing should be an integral part of system testing. Controls must be viewed as a system within a system, and tested in parallel with other system tests. Knowing that approximately 50 percent of the total development effort goes into controls, a proportionate part of testing should be allocated to evaluating the adequacy of controls.

1.10.1.2.7 Parallel Testing Techniques

In the early days of computer systems, parallel testing was one of the more popular testing techniques. However, as systems become more integrated and complex, the difficulty in conducting parallel tests increases and thus the popularity of the technique diminishes. Parallel testing is used to determine that the results of the new application are consistent with the processing of the previous application or version of the application.

Objectives

The objectives of conducting parallel testing are:

- Conduct redundant processing to ensure that the new version or application performs correctly
- Demonstrate consistency and inconsistency between two versions of the same application system

How to Use Parallel Testing

Parallel testing requires that the same input data be run through two versions of the same application. Parallel testing can be done with the entire application or with a segment of the application. Sometimes a particular segment, such as the day-to-day interest calculation on a savings account, is so complex and important that an effective method of testing is to run the new logic in parallel with the old logic.

If the new application changes data formats, then the input data will have to be modified before it can be run through the new application. This also makes it difficult to automatically check the results of processing through a tape or disk file compare. The more difficulty encountered in verifying results or preparing common input, the less attractive the parallel testing technique becomes.

Parallel Test Example

Examples of the use of parallel testing include:

- Operate a new and old version of a payroll system to determine that the paychecks from both systems are reconcilable.
- Run the old version of the application system to ensure that the operational status of the old system has been maintained in the event that problems are encountered in the new application.

When to Use Parallel Testing

Parallel testing should be used when there is uncertainty regarding the correctness of processing of the new application, and the old and new versions of the application are similar. In applications like payroll, banking, and other heavily financial applications where the results of processing are similar, even though the methods may change significantly – for example, going from batch to online banking – parallel testing is one of the more effective methods of ensuring the integrity of the new application.

1.10.2 Examples of Specific Testing Techniques

There are numerous specific testing techniques. Some can be performed using test tools. A discussion of the more commonly used specific techniques follow:

1.10.2.1 White-Box Testing

White-box testing assumes that the path of logic in a unit or program is known. White-box testing consists of testing paths, branch by branch, to produce predictable results. The following are white-box testing techniques:

- Statement Coverage

  Execute all statements at least once.

- Decision Coverage

  Execute each decision direction at least once.

- Condition Coverage

  Execute each decision with all possible outcomes at least once.

- Decision/Condition Coverage

  Execute all possible combinations of condition outcomes in each decision. Treat all iterations as two-way conditions exercising the loop zero times and one time.

- Multiple Condition Coverage

  Invoke each point of entry at least once.

Choose the combinations of techniques appropriate for the application. You can have an unmanageable number of test cases, if you conduct too many combinations of these techniques.

1.10.2.2 Black-Box Testing

Black-box testing focuses on testing the function of the program or application against its specification. Specifically, this technique determines whether combinations of inputs and operations produce expected results.

When creating black-box test cases, the input data used is critical. Three successful techniques for managing the amount of input data required include:

- Equivalence Partitioning

  An equivalence class is a subset of data that is representative of a larger class. Equivalence partitioning is a technique for testing equivalence classes rather than undertaking exhaustive testing of each value of the larger class. For example, a program which edits credit limits within a given range ($10,000 - $15,000) would have three equivalence classes:

  - < $10,000 (invalid)
  - Between $10,000 and $15,000 (valid)
  - > $15,000 (invalid)

- Boundary Analysis

  A technique that consists of developing test cases and data that focus on the input and output boundaries of a given function. In same credit limit example, boundary analysis would test:

  - Low boundary +/- one ($9,999 and $10,001)
  - On the boundary ($10,000 and $15,000)
  - Upper boundary +/- one ($14,999 and $15,001)

- Error Guessing

  Test cases can be developed based upon the intuition and experience of the tester. For example, in an example where one of the inputs is the date, a tester may try February 29, 2000.

1.10.2.3 Incremental Testing

Incremental testing is a disciplined method of testing the interfaces between unit-tested programs as well as between system components. It involves adding unit-tested programs to a given module or component one by one, and testing each resultant combination. There are two types of incremental testing:

- Top-down

  Begin testing from the top of the module hierarchy and work down to the bottom using interim stubs to simulate lower interfacing modules or programs. Modules are added in descending hierarchical order.

- Bottom-up

  Begin testing from the bottom of the hierarchy and work up to the top. Modules are added in ascending hierarchical order. Bottom-up testing requires the development of driver

modules, which provide the test input, that call the module or program being tested, and display test output.

There are pros and cons associated with each of these methods, although bottom-up testing is often thought to be easier to use. Drivers are often easier to create than stubs, and can serve multiple purposes.

1.10.2.4 Thread Testing

This test technique, which is often used during early integration testing, demonstrates key functional capabilities by testing a string of units that accomplish a specific function in the application. Thread testing and incremental testing are usually utilized together. For example, units can undergo incremental testing until enough units are integrated and a single business function can be performed, threading through the integrated components.

When testing client/server applications, these techniques are extremely critical. An example of an effective strategy for a simple two-tier client/server application could include:

1. Unit and bottom-up incremental testing of the application server components
2. Unit and incremental testing of the GUI (graphical user interface) or client components
3. Testing of the network
4. Thread testing of a valid business transaction through the integrated client, server, and network

Table 1-12 illustrates how the various techniques can be used throughout the standard test stages.

Table 1-12: Testing Techniques and Standard Test Stages

	Technique			
Stages	**White-Box**	**Black-Box**	**Incremental**	**Thread**
Unit Testing	X			
String/Integration Testing	X	X	X	X
System Testing		X	X	X
Acceptance Testing		X		

It is important to note that when evaluating the paybacks received from various test techniques, white-box or program-based testing produces a higher defect yield than the other dynamic techniques when planned and executed correctly.

1.10.2.5 Requirements Tracing

One key component of a life cycle test approach is verifying at each step of the process the inputs to a stage are correctly translated and represented in the resulting artifacts. Requirements, or stakeholder needs, are one of these key inputs that must be traced throughout the rest of the software development life cycle.

The primary goal of software testing is to prove that the user or stakeholder requirements are actually delivered in the final product developed. This can be accomplished by tracing these requirements, both functional and non-functional, into analysis and design models, class and sequence diagrams, test plans and code to ensure they're delivered. This level of traceability also enables project teams to track the status of each requirement throughout the development and test process.

1.10.2.5.1 Example

If a project team is developing an object-oriented Internet application, the requirements or stakeholder needs will be traced to use cases, activity diagrams, class diagrams and test cases or scenarios in the analysis stage of the project. Reviews for these deliverables will include a check of the traceability to ensure that all requirements are accounted for.

In the design stage of the project, the tracing will continue to design and test models. Again, reviews for these deliverables will include a check for traceability to ensure that nothing has been lost in the translation of analysis deliverables. Requirements mapping to system components drives the test partitioning strategies. Test strategies evolve along with system mapping. Test cases to be developed need to know where each part of a business rule is mapped in the application architecture. For example, a business rule regarding a customer phone number may be implemented on the client side as a GUI field edit for high performance order entry. In another it may be implemented as a stored procedure on the data server so the rule can be enforced across applications.

When the system is implemented, test cases or scenarios will be executed to prove that the requirements were implemented in the application. Tools can be used throughout the project to help manage requirements and track the implementation status of each one.

1.10.2.6 Desk Checking and Peer Review

Desk checking is the most traditional means for analyzing a program. It is the foundation for the more disciplined techniques of walkthroughs, inspections, and reviews. In order to improve the effectiveness of desk checking, it is important that the programmer thoroughly review the problem definition and requirements, the design specification, the algorithms and the code listings. In most instances, desk checking is used more as a debugging technique than a testing technique. Since seeing one's own errors is difficult, it is better if another person does the desk checking. For example, two programmers can trade listings and read each other's code. This approach still lacks the group dynamics present in formal walkthroughs, inspections, and reviews.

Another method, not directly involving testing, which tends to increase overall quality of software production, is peer review. There is a variety of implementations of peer review, but all are based on a review of each programmer's code. A panel can be set up which reviews sample code on a regular basis for efficiency, style, adherence to standard, etc., and which provides feedback to the individual programmer. Another possibility is to maintain a notebook of required "fixes" and revisions to the software and indicate the original programmer or designer. In a "chief programmer team" environment, the librarian can collect data on programmer runs, error reports, etc., and act as a review board or pass the information on to a peer review panel.

1.10.2.7 Walkthroughs, Inspections, and Reviews

Walkthroughs and inspections are formal manual techniques that are a natural evolution of desk checking. While both techniques share a common philosophy and similar organization, they are quite distinct in execution. Furthermore, while they both evolved from the simple desk check discipline of the single programmer, they are very disciplined procedures aimed at removing the major responsibility for verification from the developer.

Both procedures require a team, usually directed by a moderator. The team includes the developer, but the remaining members and the moderator should not be directly involved in the development effort. Both techniques are based on a reading of the product (e.g., requirements, specifications, or code) in a formal meeting environment with specific rules for evaluation. The difference between inspection and walkthrough lies in the conduct of the meeting. Both methods require preparation and study by the team members, and scheduling and coordination by the team moderator.

Inspection involves a step-by-step reading of the product, with each step checked against a predetermined list of criteria. These criteria include checks for historically common errors. Guidance for developing the test criteria can be found elsewhere. The developer is usually required to narrate the reading product. The developer finds many errors just by the simple act of reading aloud. Others, of course, are determined because of the discussion with team members and by applying the test criteria.

Walkthroughs differ from inspections in that the programmer does not narrate a reading of the product by the team, but provides test data and leads the team through a manual simulation of the system. The test data is walked through the system, with intermediate results kept on a blackboard or paper. The test data should be kept simple given the constraints of human simulation. The purpose of the walkthrough is to encourage discussion, not just to complete the system simulation on the test data. Most errors are discovered through questioning the developer's decisions at various stages, rather than through the application of the test data.

At the problem definition stage, walkthrough and inspection can be used to determine if the requirements satisfy the testability and adequacy measures as applicable to this stage in the development. If formal requirements are developed, formal methods, such as correctness techniques, may be applied to ensure adherence with the quality factors.

Walkthroughs and inspections should again be performed at the preliminary and detailed design stages. Design walkthroughs and inspection will be performed for each module and module interface. Adequacy and testability of the module interfaces are very important. Any changes that result from these analyses will cause at least a partial repetition of the verification at both stages and between the stages. A reexamination of the problem definition and requirements may also be required.

Finally, the walkthrough and inspection procedures should be performed on the code produced during the construction stage. Each module should be analyzed separately and as integrated parts of the finished software.

Design reviews and audits are commonly performed as stages in software development as follows:

- System Requirements Review

  This review is an examination of the initial progress during the problem definition stage and of the convergence on a complete system configuration. Test planning and test documentation are begun at this review.

- System Design Review

  This review occurs when the system definition has reached a point where major system modules can be identified and completely specified along with the corresponding test requirements. The requirements for each major subsystem are examined along with the preliminary test plans. Tools required for verification support are identified at this stage.

- Preliminary Design Review

  This review is a formal technical review of the basic design approach for each major subsystem or module. The revised requirements and preliminary design specifications for each major subsystem and all test plans, procedures and documentation are reviewed at this stage. Development and verification tools are further identified at this stage. Changes in requirements will lead to an examination of the test requirements to maintain consistency.

- Final Design Review

  This review occurs just prior to the beginning of the construction stage. The complete and detailed design specifications for each module and all draft test plans and documentation are examined. Again, consistency with previous stages is reviewed, with particular attention given to determining if test plans and documentation reflect changes in the design specifications at all levels.

- Final Review

  This review determines through testing that the final coded subsystem conforms to the final system specifications and requirements. It is essentially the subsystem acceptance test.

Three rules should be followed for all reviews:

1. The product is reviewed, not the producer.
2. Defects and issues are identified, not corrected.
3. All members of the reviewing team are responsible for the results of the review.

Reviews are conducted to utilize the variety of perspectives and talents brought together in a team. The main goal is to identify defects within the stage or phase of the project where they originate, rather than in later test stages; this is referred to as “stage containment.” As reviews are generally greater than 65 percent efficient in finding defects, and testing is often less than 30 percent efficient, the advantage is obvious. In addition, since defects identified in the review process are found earlier in the life cycle, they are less expensive to correct.

Another advantage of holding reviews is not readily measurable. That is, reviews are an efficient method of educating a large number of people on a specific product/project in a relatively short period of time. Semiformal reviews are especially good for this, and indeed, are often held for just that purpose. In addition to learning about a specific product/project, team members are exposed to a variety of approaches to technical issues, a cross-pollination effect. Finally, reviews provide training in and enforce the use of standards, as nonconformance to standards is considered a defect and reported as such.

1.10.2.8 Proof of Correctness Techniques

Proof techniques as methods of validation have been used since von Neumann's time (the mathematician, promoter of the stored program concept and developer of the von Neumann Architecture in the 1940’s). These techniques usually consist of validating the consistency of an output "assertion" (specification) with respect to a program (requirements or design specification) and an input assertion (specification). In the case of software programs, the assertions are statements about the software program's variables. The software program is "proved" if, whenever the input assertion is true for particular values of variables, and the program executes, then it can be shown that the output assertion is true for the program's variables that have changed. The issue of abnormal termination is normally treated separately.

There are two approaches to proof of correctness: formal proof and informal proof. A formal proof consists of developing a mathematical logic consisting of axioms and inference rules and defining a proof either to be a proof tree in the natural deduction style or to be a finite sequence of axioms and inference rules in the Hilbert-Ackermann style. The statement to be proved is at the root of the proof tree or is the last object in the proof sequence. Since the formal proof logic must also "talk about" the domain of the program and the operators that occur in the program, a second mathematical logic must be employed. This second mathematical logic is usually not decidable.

Most recent research in applying proof techniques to verification has concentrated on programs. The techniques apply, however, equally well to any level of the development life cycle where a formal representation or description exists.

Informal proof techniques follow the logical reasoning behind the formal proof techniques but without the formal logical system. Often the less formal techniques are more palatable to the

programmers. The complexity of informal proof ranges from simple checks such as array bounds not being exceeded, to complex logic chains showing noninterference of processes accessing common data. Programmers always use informal proof techniques implicitly. To make them explicit is similar to imposing disciplines, such as structured walkthroughs, on the programmer.

1.10.2.9 Simulation

Simulation is used in real-time systems development where the "real-world" interface is critical and integration with the system hardware is central to the total design. In many non-real-time applications, simulation is a cost effective verification and test-data generation technique.

To use simulation as a verification tool several models must be developed. Verification is performed by determining if the model of the software behaves as expected on models of the computational and external environments using simulation. This technique also is a powerful way of deriving test data. Inputs are applied to the simulated model and the results recorded for later application to the actual code. This provides an "oracle" for testing. The models are often "seeded" with errors to derive test data, which distinguish these errors. The data sets derived cause errors to be isolated and located as well as detected during the testing phase of the construction and integration stages.

To develop a model of the software for a particular stage in the development life cycle a formal representation compatible with the simulation system is developed. This may consist of the formal requirement specification, the design specification, or separate model of the program behavior. If a different model is used, then the developer will need to demonstrate and verify that the model is a complete, consistent, and accurate representation of the software at the stage of development being verified.

The next steps are to develop a model of the computational environment in which the system will operate, a model of the hardware on which the system will be implemented, and a model of the external demands on the total system. These models can be largely derived from the requirements, with statistical representations developed for the external demand and the environmental interactions. The software behavior is then simulated with these models to determine if it is satisfactory.

Simulating the system at the early development stages is the only means of determining the system behavior in response to the eventual implementation environment. At the construction stage, since the code is sometimes developed on a host machine quite different from the target machine, the code may be run on a simulation of the target machine under interpretive control.

Simulation also plays a useful role in determining the performance of algorithms. While this is often directed at analyzing competing algorithms for cost, resource, or performance trade-offs, the simulation under real loads does provide error information.

1.10.2.10 Boundary Value Analysis

The problem of deriving test data sets is to partition the program domain in some meaningful way so that input data sets, which span the partition, can be determined. There is no direct, easily stated procedure for forming this partition. It depends on the requirements, the program domain, and the creativity and problem understanding of the programmer. This partitioning, however, should be performed throughout the development life cycle.

At the requirements stage a coarse partitioning is obtained according to the overall functional requirements. At the design stage, additional functions are introduced which define the separate modules allowing for a refinement of the partition. Finally, at the coding stage, submodules implementing the design modules introduce further refinements. The use of a top down testing methodology allows each of these refinements to be used to construct functional test cases at the appropriate level.

Once the program domain is partitioned into input classes, functional analysis can be used to derive test data sets. Test data should be chosen which lie both inside each input class and at the boundary of each class. Output classes should also be covered by input that causes output at each class boundary and within each class. These data are the extremal and non-extremal test sets. Determination of these test sets is often called boundary value analysis or stress testing.

1.10.2.11 Error Guessing and Special Value Analysis

Some people have a natural intuition for test data generation. While this ability cannot be completely described nor formalized, certain test data seem highly probable to catch errors. Zero input values and input values that cause zero outputs are examples of where a tester may guess an error could occur. Guessing carries no guarantee for success, but neither does it carry any penalty.

1.10.2.12 Cause-Effect Graphing

Cause-effect graphing is a technique for developing test cases for programs from the high-level specifications. A high-level specification of requirements states desired characteristics of behavior for the system. These characteristics can be used to derive test data. Problems arise, however, of a combinatorial nature. For example, a program that has specified responses to eight characteristic stimuli (called causes) given some input has potentially 256 "types" of input (i.e., those with characteristics 1 and 3, those with characteristics 5, 7, and 8, etc.). A naive approach to test case generation would be to try to generate all 256 types. A more methodical approach is to use the program specifications to analyze the program's effect on the various types of inputs.

The program's output domain can be partitioned into various classes called effects. For example, inputs with characteristic 2 might be subsumed by those with characteristics 3 and 4. Hence, it would not be necessary to test inputs with just characteristic 2 and also inputs with

characteristics 3 and 4, for they cause the same effect. This analysis results in a partitioning of the causes according to their corresponding effects.

A limited entry decision table is then constructed from the directed graph reflecting these dependencies (i.e., causes 2 and 3 result in effect 4, causes 2, 3 and 5 result in effect 6, etc.). The decision table is then reduced and test cases chosen to exercise each column of the table. Since many aspects of the cause-effect graphing can be automated, it is an attractive tool for aiding in the generation of functional test cases.

1.10.2.13 Design-Based Functional Testing

The techniques described above derive test data sets from analysis of functions specified in the requirements. Functional analysis can be extended to functions used in the design process. A distinction can be made between requirements functions and design functions. Requirements functions describe the overall functional capabilities of a program. In order to implement a requirements function it is usually necessary to invent other "smaller functions." These other functions are used to design the program. If one thinks of this relationship as a tree structure, then a requirements function would be represented as a root node. All functional capabilities represented by boxes at the second level in the tree correspond to design functions. The implementation of a design function may require the invention of other design functions.

To utilize design-based functional testing, the functional design trees as described above are constructed. The trees document the functions used in the design of the program. The functions included in the design trees must be chosen carefully. The most important selection feature is that the function be accessible for independent testing. It must be possible to apply the appropriate input values to test the function, to derive the expected values for the function, and to observe the actual output computed by the code implementing the function.

Each of the functions in the functional design tree, if top down design techniques are followed, can be associated with the final code used to implement that function. This code may consist of one or more procedures, parts of a procedure, or even a single statement. Design-based functional testing requires that the input and output variables for each design function be completely specified.

1.10.2.14 Coverage-Based Testing

Most coverage metrics are based on the number of statements, branches, or paths in the program, which are exercised by the test data. Such metrics can be used both to evaluate the test data and to aid in the generation of the test data.

Any program can be represented by a graph. The nodes represent statements or collections of sequential statements. Directed lines or edges that connect the nodes represent the control flow. A node with a single exiting edge to another node represents a sequential code segment. A node with multiple exiting edges represents a branch predicate or a code segment containing a branch predicate as the last statement.

On a particular set of data, a program will execute along a particular path, where certain branches are taken or not taken depending on the evaluation of branch predicates. Any program path can be represented by a sequence, possibly with repeating subsequences (when the program has backward branches), of edges from the program graph. These sequences are called path expressions. Each path or each data set may vary depending on the number of loop iterations caused. A program with variable loop control may have effectively an infinite number of paths. Hence, there are potentially an infinite number of path expressions.

To completely test the program structure, the test data chosen should cause the execution of all paths. Since this is not possible in general, metrics have been developed which give a measure of the quality of test data based on the proximity to this ideal coverage. Path coverage determination is further complicated by the existence of infeasible paths. Often a program has been inadvertently designed so that no data will cause the execution of certain paths. Automatic determination of infeasible paths is generally difficult if not impossible. A main theme in structured top-down design is to construct modules that are simple and of low complexity so that all paths, excluding loop iteration, may be tested and that infeasible paths may be avoided.

All techniques for determining coverage metrics are based on graph representations of programs. Varieties of metrics exist ranging from simple statement coverage to full path coverage. There have been several attempts to classify these metrics; however, new variations appear so often that such attempts are not always successful. We will discuss the major ideas without attempting to cover all the variations.

The simplest metric measures the percentage of statements executed by all the test data. Since coverage tools supply information about which statements have been executed (in addition to the percentage of coverage), the results can guide the selection of test data to ensure complete coverage. To apply the metric, the program or module is instrumented by hand or by a preprocessor. A post processor or manual analysis of the results reveals the level of statement coverage. Determination of an efficient and complete test data set satisfying this metric is more difficult. Branch predicates that send control to omitted statements should be examined to help determine input data that will cause execution of omitted statements.

A slightly stronger metric measures the percentage of segments executed under the application of all test data. A segment in this sense corresponds to a decision-to-decision path (dd-path). It is a portion of a program path beginning with the execution of a branch predicate and including all statements up to the evaluation (but not execution) of the next branch predicate. Segment coverage guarantees statement coverage. It also covers branches with no executable statements; e.g., an IF-THEN-ELSE with no ELSE statement still requires data causing the predicate to be evaluated as both True and False. Techniques similar to those needs for statement coverage are used for applying the metric and deriving test data.

The next logical step is to strengthen the metric by requiring separate coverage for both the exterior and interior of loops. Segment coverage only requires that both branches from a branch predicate be taken. For loops, segment coverage can be satisfied by causing the loop to be executed one or more times (interior test) and then causing the loop to be exited (exterior test). Requiring that all combinations of predicate evaluations be covered requires that each

loop be exited without interior execution for at least one data set. This metric requires more paths to be covered than segment coverage requires. Two successive predicates will require at least four sets of test data to provide full coverage. Segment coverage can be satisfied by two tests, while statement coverage may require only one test for two successive predicates.

1.10.2.15 Complexity-Based Testing

Several complexity measures have been proposed recently. Among these are cyclomatic complexity and software complexity measures. These, and many other metrics are designed to analyze the complexity of software systems and, although these are valuable new approaches to the analysis of software, many are not suitable, or have not been applied to the problem of testing. The McCabe metrics are the exception.

McCabe actually proposed three metrics: cyclomatic, essential, and actual complexity. All three are based on a graphical representation of the program being tested. The first two are calculated from the program graph, while the third is a runtime metric.

McCabe uses a property of graph theory in defining cyclomatic complexity. There are sets of linearly independent program paths through any program graph. A maximum set of these linearly independent paths, called a basis set, can always be found. Intuitively, since the program graph and any path through the graph can be constructed from the basis set, the size of this basis set should be related to the program complexity.

1.10.2.16 Statistical Analyses and Error Seeding

The most common type of test data analysis is statistical. An estimate of the number of errors in a program can be obtained from an analysis of the errors uncovered by the test data. In fact, as we shall see, this leads to a dynamic testing technique.

Let us assume that there are some number of errors, E, in the software being tested. There are two things we would like to know – a maximum estimate for the number of errors, and a level of confidence measure on that estimate. The technique is to insert known errors in the code in some way that is statistically similar to the actual errors. The test data is then applied and errors uncovered are determined. If one assumes that the statistical properties of the seeded and original errors is the same and that the testing and seeding are statistically unbiased, then

estimate $E = IS/K$

where S is the number of seeded errors, K is the number of discovered seeded errors, and I is the number of discovered unseeded errors. This estimate obviously assumes that the proportion of undetected errors is very likely to be the same for the seeded and original errors.

1.10.2.17 Mutation Analysis

Another metric is called mutation analysis. This method rests on the competent programmer hypothesis that states that a program written by a competent programmer will be, after debugging and testing, "almost correct." The basic idea of the method is to seed the program

to be tested with errors, creating several mutants of the original program. The program and its mutants are then run interpretively on the test set. If the test set is adequate, it is argued, it should be able to distinguish between the program and its mutants.

The method of seeding is crucial to the success of the technique and consists of modifying single statements of the program in a finite number of "reasonable" ways. The developers conjecture a coupling effect that implies that these "first order mutants" cover the deeper, more subtle errors that might be represented by higher order mutants. The method has been subject to a small number of trials and so far has been successfully used interactively to develop adequate test data sets. It should be noted that the method derives both branch coverage and statement coverage metrics as special cases.

It must be stressed that mutation analysis, and its appropriateness, rests on the competent programmer and coupling effect theses. Since neither is provable, they must be empirically demonstrated to hold over a wide variety of programs before the method of mutations can itself be validated.

1.10.2.18 Flow Analysis

Data and control flow analysis are similar in many ways. Both are based upon graphical representation. In control flow analysis, the program graph has nodes that represent a statement or segment possibly ending in a branch predicate. The edges represent the allowed flow of control from one segment to another. The control flow graph is used to analyze the program behavior, to locate instrumentation breakpoints, to identify paths, and in other static analysis activities. In data flow analysis, graph nodes usually represent single statements, while the edges still represent the flow of control. Nodes are analyzed to determine the transformations made on program variables. Data flow analysis is used to discover program anomalies such as undefined or unreferenced variables.

In data flow analysis, we are interested in tracing the behavior of program variables as they are initialized and modified while the program executes. This behavior can be classified by when a particular variable is referenced, defined, or undefined in the program. A variable is referenced when its value must be obtained from memory during the evaluation of an expression in a statement. For example, a variable is referenced when it appears on the right-hand side of an assignment statement, or when it appears as an array index anywhere in a statement. A variable is defined if a new value for that variable results from the execution of a statement, such as when a variable appears on the left-hand side of an assignment. A variable is unreferenced when its value is no longer determinable from the program flow. Examples of unreferenced variables are local variables in a subroutine after exit and DO indices on loop exit.

Data flow analysis is performed by associating, at each node in the data flow graph, values for tokens (representing program variables) which indicate whether the corresponding variable is referenced, unreferenced, or defined with the execution of the statement represented by that node.

1.10.2.19 Symbolic Execution

Symbolic execution is a method of symbolically defining data that forces program paths to be executed. Instead of executing the program with actual data values, the variable names that hold the input values are used. Thus all variable manipulations and decisions are made symbolically. Consequently, all variables become string variables, all assignments become string assignments and all decision points are indeterminate.

The result of a symbolic execution is a large, complex expression. The expression can be decomposed and viewed as a tree structure where each leaf represents a path through the program. The symbolic values of each variable are known at every point within the tree and the branch points of the tree represent the decision points of the program. Every program path is represented in the tree, and every branch path is effectively taken.

If the program has no loops, then the resultant tree structure is finite. The tree structure can then be used as an aid in generating test data that will cause every path in the program to be executed. The predicates at each branch point of the tree structure for a particular path are then collected into a conjunction. Data that causes a particular path to be executed can be found by determining which data will make the path conjunction true. If the predicates are equalities, inequalities, and orderings, the problem of data selection becomes the classic problem of trying to solve a system of equalities and orderings.

There are two major difficulties with using symbolic execution as a test set construction mechanism. The first is the combinatorial explosion inherent in the tree structure construction. The number of paths in the symbolic execution tree structure may grow as an exponential in the length of the program leading to serious computational difficulties. If the program has loops, then the symbolic execution tree structure is necessarily infinite. Usually only a finite number of loop executions is required enabling a finite loop unwinding to be performed. The second difficulty is that the problem of determining whether the conjunct has values that satisfy it is undecidable even with restricted programming languages. For certain applications, however, the method has been successful.

Another use of symbolic execution techniques is in the construction of verification conditions from partially annotated programs. Typically, the program has attached to each of its loops an assertion, called an invariant, which is true at the first statement of the loop and at the last statement of the loop (thus the assertion remains “invariant” over one execution of the loop). From this assertion, an assertion true before entrance to the loop and assertions true after exit of the loop can be constructed. The program can then be viewed as "free" of loops (i.e., each loop is considered as a single statement) and assertions extended to all statements of the program (so it is fully annotated) using techniques similar to the backward substitution method described above for symbolic execution.

1.10.3 Combining Specific Testing Techniques

There are many ways in which the techniques described in this skill category can be used in concert to form more powerful and efficient testing techniques. One of the more common

combinations today is the merger of standard testing techniques with formal verification. Our ability, through formal methods, to verify significant segments of code is improving, and moreover there are certain modules, which for security or reliability reasons, justify the additional expense of formal verification.

Other possibilities include the use of symbolic execution or formal proof techniques to verify segments of code, which through coverage analysis have been shown to be most frequently executed. Mutation analysis, for some special cases like decision tables, can be used to fully verify programs. Formal proof techniques may be useful in one of the problem areas of mutation analysis, the determination of equivalent mutants.

Osterweil addresses the issue of how to combine efficiently powerful techniques in one systematic method (combining dataflow analysis, symbolic execution, elementary theorem proving, dynamic assertions, and standard testing). As has been mentioned, symbolic execution can be used to generate dynamic assertions. Here, paths are executed symbolically so that each decision point and every loop has an assertion. The assertions are then checked for consistency using both dataflow and proof techniques. If all the assertions along the path are consistent, they can be reduced to a single dynamic assertion for the path. Theorem proving techniques can be employed to "prove" the path assertion and termination, or the path can be tested and the dynamic assertions evaluated for the test data.

The technique allows for several trade-offs between testing and formal methods. For instance, symbolically derived dynamic assertions are more reliable than manually derived assertions, but cost more to generate. Consistency analysis of the assertions using proof and dataflow techniques adds cost at the front end, but reduces the execution overhead. Finally there is the obvious trade-off between theorem proving and testing to verify the dynamic assertions.

Skill Category

2

Building the Test Environment

The test environment is comprised of all the conditions, circumstances, and influences surrounding and affecting the testing of software. The environment includes the organization's policies, procedures, culture, attitudes, rewards, test processes, test tools, methods for developing and improving test processes, management's support of software testing, as well as any test labs developed for the purpose of testing software and multiple operating environments. This skill category also includes the process for assuring the test environment fairly represents the production environment to enable realistic testing to occur.

2.1 *Management Support*	**2-1**
2.2 *Test Work Processes*	**2-8**
2.3 *Test Tools*	**2-36**
2.4 *Testers Competency*	**2-49**

2.1 Management Support

Without adequate management support testing is rarely performed effectively. For example, if management views development as a more important activity than testing, they will spend more of their personal time with developers and thus send the message of minimal support for testing. Management support also means that the appropriate resources will be spent on training testers and providing the processes and tools needed for testing. Management support will be discussed in these two areas:

- Management "tone"

  Management sets the tone by providing testers the resources and management time needed to do their job effectively.

- Test process alignment

  Management creates a test function that is aligned with the business needs of the organization.

2.1.1 Management Tone

Management "tone" is representative of the environment that management has established that influence the way testers work. Before explaining the control environment, testers need to know three things about the control environment, which are:

1. The control environment is established by the highest levels of management and works downward through the organization.
2. The test function cannot create the organization's control environment, but can influence how that environment is implemented within the test function.
3. The control environment will influence the way in which testers perform the work which may be ethical or unethical.

The control environment has a pervasive influence on the way test activities are structured, objectives established, and risks assessed. It also influences test activities, information and communication systems, and monitoring activities. This is true not only of their design, but also the way they work day to day. The control environment is influenced by the IT organization's history and culture. It influences the control consciousness of its people.

Effectively controlled entities strive to have competent people, instill an attitude of integrity and control consciousness, and set a positive "tone at the top." They establish appropriate policies and procedures, often including a written code of conduct, which foster shared values and teamwork in pursuit of the entity's objectives.[1]

The control environment encompasses factors discussed below. Although all are important, the extent to which each is addressed will vary with the entity. For example, the Test Manager with a small workforce and centralized operations may not establish formal lines of responsibility and detailed operating policies, but could still have an appropriate control environment.

1. The Control Environment factors are taken from The Committee of Sponsoring Organizations of the Treadway Commission (COSO) report *Internal Control – Integrated Framework*, 1992.

2.1.2 Integrity and Ethical Values

An entity's objectives and the way they are achieved are based on preferences, value judgments and management styles. Those preferences and value judgments, which are translated into standards of behavior, reflect management's integrity and its commitment to ethical values.

The effectiveness of internal controls cannot rise above the integrity and ethical values of the people, who create, administer and monitor them. Integrity and ethical values are essential elements of the control environment, affecting the design, administration and monitoring of the internal control components.

Integrity is a prerequisite for ethical behavior in all aspects of an enterprise's activities. A strong corporate ethical climate at all levels is vital to the well-being of the corporation, all of its constituencies, and the public at large. Such a climate contributes importantly to the effectiveness of company policies and control systems, and helps influence behavior that is not subject to even the most elaborate system of controls.

Establishing ethical values is often difficult because of the need to consider the concerns of several parties. Top management's values must balance the concerns of the enterprise, its employees, suppliers, customers, competitors and the public. Balancing these concerns can be a complex and frustrating effort because interests often are at odds. For example, testers may want the most current test tool, but management does not want unproven technology.

Managers of well-run enterprises have increasingly accepted the view that "ethics pays" – that ethical behavior is good business. Positive and negative examples abound. The well-publicized handling of a crisis by a pharmaceutical company involving tampering with one of its major products was both sound ethics and sound business. The impact on customer relations or stock prices of slowly leaked bad news, such as profit shortfalls or illegal acts, generally is worse than if full disclosures are made as quickly as possible.

Focusing solely on short-term results can hurt even in the short term. Concentration on the bottom line – sales or profit at any cost – often evokes unsought actions and reactions. High-pressure sales tactics, ruthlessness in negotiations or implicit offers of kickbacks, for instance, may evoke reactions that can have immediate (as well as lasting) effects.

Ethical behavior and management integrity are a product of the "corporate culture." Corporate culture includes ethical and behavioral standards, how they are communicated and how they are reinforced in practice. Official policies specify what management wants to happen. Corporate culture determines what actually happens, and which rules are obeyed, bent or ignored. Top management – starting with the CEO – plays a key role in determining the corporate culture. The CEO usually is the dominant personality in an organization, and individually often sets its ethical tone.

2.1.2.1 Incentives and Temptations

Individuals may engage in dishonest, illegal or unethical acts simply because their organizations give them strong incentives or temptations to do so. Emphasis on "results," particularly in the short term, fosters an environment in which the price of failure becomes very high.

Incentives cited for engaging in fraudulent or questionable financial reporting practices and, by extension, other forms of unethical behavior are:

- Pressure to meet unrealistic performance targets, particularly for short-term results
- High performance-dependent rewards.

There may be "temptations" for employees to engage in improper acts:

- Nonexistent or ineffective controls, such as poor segregation of duties in sensitive areas that offer temptations to steal or to conceal poor performance.
- High decentralization that leaves top management unaware of actions taken at lower organizational levels and thereby reduces the chances of getting caught.
- A weak internal audit function that does not have the ability to detect and report improper behavior.
- Penalties for improper behavior that are insignificant or unpublicized and thus lose their value as deterrents.

Removing or reducing these incentives and temptations can go a long way toward diminishing undesirable behavior. As suggested, this can be achieved following sound and profitable business practices. For example, performance incentives – accompanied by appropriate controls – can be a useful management technique as long as the performance targets are realistic. Setting realistic performance targets is a sound motivational practice; it reduces counterproductive stress as well as the incentive for fraudulent financial reporting that unrealistic targets create. Similarly, a well-controlled reporting system can serve as a safeguard against temptation to misstate performance.

2.1.2.2 Providing and Communicating Moral Guidance

The most effective way of transmitting a message of ethical behavior throughout the organization is by example. People imitate their leaders. Employees are likely to develop the same attitudes about what's right and wrong – and about internal control – as those shown by top management. Knowledge that the CEO has "done the right thing" ethically when faced with a tough business decision sends a strong message to all levels of the organization.

Setting a good example is not enough. Top management should verbally communicate the entity's values and behavioral standards to employees. A formal code of corporate conduct is "a widely used method of communicating to employees the company's expectations about duty and integrity." Codes address a variety of behavioral issues, such as integrity and ethics, conflicts of interest, illegal or otherwise improper payments, and anti-competitive arrangements. Spurred in part by revelations of scandals in the defense industry, many companies have adopted such codes in recent years, along with necessary communications

channels and monitoring. While codes of conduct can be helpful, they are not the only way to transmit an organization's ethical values to employees, suppliers and customers.

Existence of a written code of conduct, and even documentation that employees received and understand it, does not ensure that it is being followed. Compliance with ethical standards, whether or not embodied in a written code of conduct, is best ensured by top management's actions and examples. Of particular importance are resulting penalties to employees who violate such codes, mechanisms that exist to encourage employee reporting of suspected violations, and disciplinary actions against employees who fail to report violations. Messages sent by management's actions in these situations quickly become embodied in the corporate culture.

2.1.3 Commitment to Competence

Competence should reflect the knowledge and skills needed to accomplish tasks that define the individual's job. How well these tasks need to be accomplished generally is a management decision that should be made considering the entity's objectives and management's strategies and plans for achievement of the objectives. There often is a trade-off between competence and cost – it is not necessary, for instance, to hire a computer science major to operate a computer.

Management needs to specify the competence levels for particular jobs and to translate those levels into requisite knowledge and skills. The necessary knowledge and skills may in turn depend on individual's intelligence, training and experience. Among the many factors considered in developing knowledge and skill levels are the nature and degree of judgment to be applied to a specific job. There often can be a trade-off between the extent to supervision and the requisite competence level of the individual.

2.1.4 Management's Philosophy and Operating Style

Management's philosophy and operating style affect the way testing is managed, including the kinds of business risks accepted. An entity that has been successful taking significant risks may have a different outlook on internal control than one that has faced harsh economic or regulatory consequences as a result of venturing into dangerous territory. An informally managed company may control operations largely by face-to-face contact with key managers. A more formally managed one may rely more on written policies, performance indicators and exception reports.

Other elements of management's philosophy and operating style include attitudes toward financial reporting, conservative or aggressive selection from available alternative accounting principles, conscientiousness and conservatism with which accounting estimates are developed, and attitudes toward data processing and accounting functions and personnel.

2.1.5 Organizational Structure

An entity's organizational structure provides the framework within which its activities for achieving entity-wide objectives are planned, executed, controlled and monitored. Activities may relate to what is sometimes referred to as the value chain: inbound (requirements) activities, operations or production, outbound (software), deployment and maintenance. There may be support functions, relating to administration, human resources or technology development.

Significant aspects of establishing a relevant organizational structure include defining key areas of authority and responsibility and establishing appropriate lines of reporting. An entity develops an organizational structure suited to its needs. Some are centralized, others decentralized. Some have direct reporting relationships, others are more of a matrix organization. Some entities are organized by industry or product line, by geographical location or by a particular distribution or marketing network. Other entities, including many state and local governmental units and not-for-profit institutions, are organized on a functional basis.

The appropriateness of an entity's organizational structure depends, in part, on its size and the nature of its activities. A highly structured organization, including formal reporting lines and responsibilities, may be appropriate for a large entity with numerous operating divisions, including foreign operations. However, it could impede the necessary flow of information in a small entity. Whatever the structure, an entity's activities will be organized to carry out the strategies designed to achieve particular objectives.

2.1.5.1 Assignment of Authority and Responsibility

Management has an important function with the assignment of authority and responsibility for operating activities, and establishment of reporting relationships and authorization protocols. It involves the degree to which individuals and teams are encouraged to use initiative in addressing issues and solving problems, as well as limits of their authority. This management function also deals with policies describing appropriate business practices, knowledge and experience of key personnel, and resources provided for carrying out duties.

There is a growing tendency to push authority downward to bring decision-making closer to front-line personnel. An organization may take this tact to become more market-driven or quality focused – perhaps to eliminate defects, reduce cycle time or increase customer satisfaction. To do so, the organization needs to recognize and respond to changing priorities in market opportunities, business relationships, and public expectations.

Alignment of authority and accountability often is designed to encourage individual initiatives, within limits. Delegation of authority, or "empowerment," means surrendering central control of certain business decisions to lower echelons – to the individuals who are closest to everyday business transactions. This may involve empowerment to sell products at discount prices; negotiate long-term supply contracts, licenses or patents; or enter alliances or joint ventures.

A critical challenge is to delegate only to the extent required to achieve objectives. This requires ensuring that risk acceptance is based on sound practices for identification and minimization of risk, including sizing risks and weighing potential losses versus gains in arriving at good business decisions.

Another challenge is ensuring that all personnel understand the entity's objectives. It is essential that each individual know how his or her actions interrelate and contribute to achievement of the objectives.

Increased delegation sometimes is accompanied by, or the result of, streamlining or "flattening" of an entity's organizational structure, and is intentional. Purposeful structural change to encourage creativity, initiative and the capability to react quickly can enhance competitiveness and customer satisfaction. Such increased delegation may carry an implicit requirement for a higher level of employee competence, as well as greater accountability. It also requires effective procedures for management to monitor results.

The control environment is greatly influenced by the extent to which individuals recognize that they will be held accountable. This holds true all the way to the chief executive, who has ultimate responsibility for all activities within an entity, including the internal control system.

2.1.5.2 Human Resource Policies and Practices

Human resource practices send messages to employees regarding expected levels of integrity, ethical behavior and competence. Such practices relate to hiring, orientation, training, evaluating, counseling, promoting, compensating and remedial actions:

- Standards for hiring the most qualified individuals, with emphasis on educational background, prior work experience, past accomplishments and evidence of integrity and ethical behavior, demonstrate an organization's commitment to competent and trustworthy people.
- Recruiting practices that include formal, in-depth employment interviews and informative and insightful presentations on the organization's history, culture and operating style send a message that the organization is committed to its people.
- Training policies that communicate prospective roles and responsibilities and include practices such as training schools and seminars, simulated case studies and role-play exercises, illustrate expected levels of performance and behavior.
- Rotation of personnel and promotions driven by periodic performance appraisals demonstrate the entity's commitment to the advancement of qualified personnel to higher levels of responsibility.
- Competitive compensation programs that include bonus incentives serve to motivate and reinforce outstanding performance. Disciplinary actions send a message that violations of expected behavior will not be tolerated.

It is essential that personnel be equipped for new challenges as issues that organizations face change and become more complex – driven in part by rapidly changing technologies and increasing competition. Education and training – whether classroom instruction, self-study or on-the-job training – must prepare an entity's people to keep pace and deal effectively with the

evolving environment. They will also strengthen the organization's ability to affect quality initiatives. Hiring of competent people and one-time training are not enough; the education process must be ongoing.

2.2 Test Work Processes

Work processes are the policies, standards and procedures in a quality IT environment. Test work processes are those work processes specific to the testing activity. Once management commits to create a work environment conducive to effective software testing, test work processes need to be created. It is the tester's responsibility to follow these test work processes; and management's responsibility to change the processes if the processes do not work. A process to improve the test work processes should be implemented to improve the effectiveness and efficiency of the policies, standards, and procedures.

It is important to have a common understanding of the following definitions:

- Policy

  Managerial desires and intents concerning either process (intended objectives) or products (desired attributes).

- Standards

  The measure used to evaluate products and identify nonconformance. The basis upon which adherence to policies is measured.

- Procedure

  The step-by-step method that ensures that standards are met.

Policies provide direction, standards are the rules or measures by which the implemented policies are measured, and the procedures are the means used to meet or comply with the standards. These definitions show the policy at the highest level, standards established next, and procedures last. However, the worker sees a slightly different view, which is important in explaining the practical view of standards.

The following aspects of test work processes are critical to the success of the testing activity and are discussed further:

- The importance of work processes
- The responsibilities

2.2.1 The Importance of Work Processes

It is important for a quality IT environment to establish, adhere to, and maintain work processes in the testing activity. It is also critical that the work processes represent sound policies, standards and procedures. It must be emphasized that the purposes and advantages to

standards discussed below exist only when sound work processes are in place. If the processes are defective or out of date, the purposes will not be met. Poor standards can, in fact, impede quality and reduce productivity. Thus, constant attention is needed to operate and improve an organization's standards program.

The major purposes for and advantages to having work processes for testers are:

- Improves communication

  The standards define the products to be produced. Not only are the products defined, but also the detailed attributes of each product are defined. This definition attaches names to the products and the attributes of the products. Thus, in a standardized environment when someone says, "a requirements document," it is clear what that means and what attributes must be defined in order to have requirements defined. In an environment without standards, the communication between workers is reduced because when one says requirements, another is probably not certain what that means.

- Enables knowledge transfer

  Processes are in fact "expert systems." They enable the transfer of knowledge from the more senior people to the more junior. Once a procedure has been learned, it can be formalized and all people in the department can perform that procedure with reasonable effort. In an environment in which processes are not well defined, some people may be able to do a job very effectively, and others perform the same job poorly. The formalization of process engineering should raise the productivity of the poor performers in the department and at the same time not hinder the productivity of the effective performers.

- Improves productivity

  It is difficult to improve productivity throughout a function without first standardizing how the function is performed. Once it is clear to everyone how the function is performed, they can contribute to improve that process. The key to productivity becomes constant improvement. Since no one person can do everything the best, each shortcut or better method identified by anyone in the organization can be quickly incorporated into the procedures and the benefits gained by everybody.

- Assists with mastering new technology

  The processes are designed to help the IT function master new technology. There is a huge cost of learning whenever new technology is introduced. Each individual must invest time and energy into mastering new technology. Effective processes assist in that technological mastery. It is knowledge transfer for new technology.

- Reduces defects and cost

  It costs a lot of money to make defective products. If workers make defects through lack of mastery of technology or using ineffective processes, the organization has to pay for not only making the defect, but also for searching and correcting it. Doing it right the first time significantly reduces the cost of doing work.

2.2.2 Developing Work Processes

Prior to creating any processes, the Process Engineering Committee, as discussed in 2.2.4.4.3 *Role of the Process Engineering Committee*, must develop a standard and procedure for developing standards and procedures. The standards and procedures developed by the standards program are products. Those products must be standardized just as the other products produced in an IT function. After all, a standard is an attribute of a product. Therefore, until the standard and procedure development process has been standardized, the same type of process the Process Engineering Committee is proposing the IT staff follow in creating their products cannot develop it.

2.2.2.1 Defining the Attributes of a Standard for a Standard

The standard for a standard defines the format, style, and attributes of a document called a standard. The standard for a procedure defines the format, style, and attributes of a document called a procedure. The standard for a standard and standard for a procedure become the prototype that will be used by the Ad Hoc Committees in developing standards and accompanying procedure(s). Thus, it is an important document and warrants the full attention of the Process Engineering Committee. It is, in fact, the only standard and procedure developed by the Process Engineering Committee and Process Engineering Manager. They, too, may wish to form an Ad Hoc Committee to assist them in this effort.

2.2.2.2 Developing a Test Standard

When developing a test standard, one should keep in mind the purpose of a test standard. A test standard should define the following:

- Testing policies
- Testing standards
- Testing procedures (Do Procedures)
- Testing procedures (Check Procedures)

2.2.2.2.1 Establishing a Testing Policy

A testing policy is management's objective for testing. It is the objective to be accomplished. A process must be in place to determine how that policy will be achieved. A workbench is one means to illustrate how a policy will be achieved. A sample testing policy is illustrated in Figure 2-1.

Testing Policy
ABC IT Department

Testing should determine that all computer systems meet stated requirements and user needs.

Figure 2-1 Simplistic Testing Policy

Good testing does not just happen, it must be planned, and a testing policy should be the cornerstone of that plan. Figure 2-1 is a simplistic testing policy that an IT department could adopt. A good practice is for management to establish the testing policy for the IT department, then have all members of IT management sign that policy as their endorsement and intention to enforce that testing policy, and then prominently display that endorsed policy where everyone in the IT department can see it.

IT management normally assumes that their staff understands the testing function and what they, management, want from testing. Exactly the opposite is normally true. Testing is not clearly defined, nor is management's intent made known regarding their desire for the type and extent of testing.

IT departments frequently adopt testing tools such as a test data generator, make the system programmer/analyst aware of those testing tools, and then leave it to the discretion of the staff how testing is to occur and to what extent. In fact, many "anti-testing" messages may be directly transmitted from management to staff. For example, pressure to get projects done on time and within budget is an anti-testing message from management. The message says, "I don't care how you get the system done, but get it done on time and within budget," which translates to the average system programmer/analyst as "Get it in on time even if it isn't tested."

2.2.3 Tester's Workbench

We need to revisit the tester's workbench in order to put the definitions into perspective. The tester's workbench, illustrated in Figure 2-2, shows that input products drive the workbench, which uses procedures and standards to produce output products.

It is the policies that define the objective for the workbenches. They may not define them directly for low-level workbenches, but will define them indirectly by establishing the areas of activity and direction for the activity. For example, if the policies define an activity of support for end users, it is normally necessary to define and establish a help desk workbench.

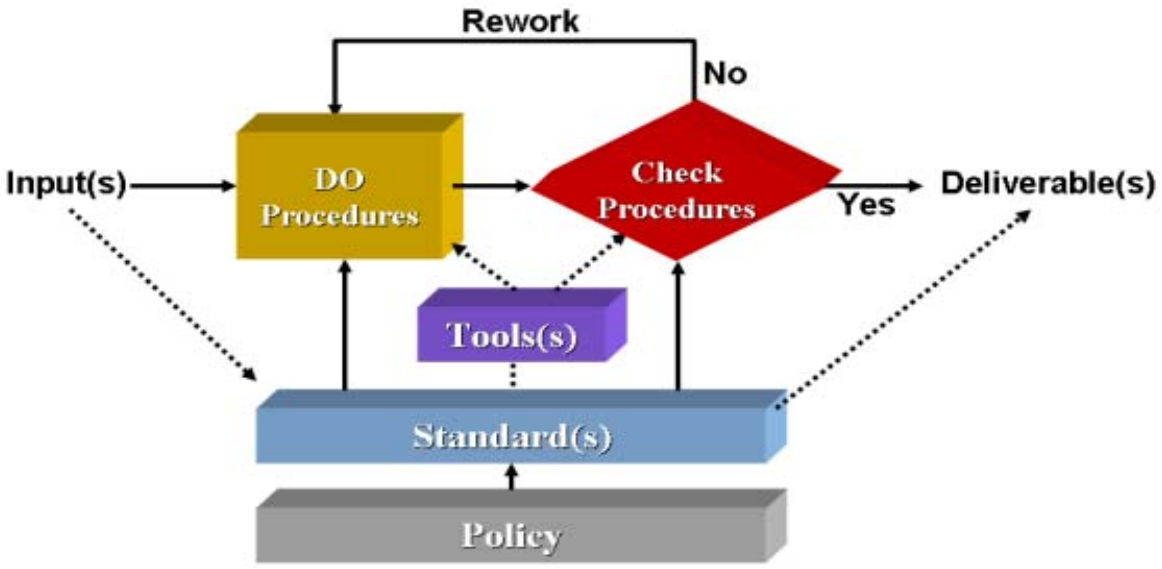


Figure 2-2 Tester's Workbench

Many of the productivity and quality problems within the test function are attributable to the incorrect or incomplete definition of tester's workbenches. For example, workbenches may not be established, or too few may be established. A test function may only have one workbench for software test planning, when in fact, they should have several; such as, a budgeting workbench, and scheduling workbench, a risk assessment workbench, and a tool selection workbench. In addition, they may have an incompletely defined test data workbench that leads to poorly defined test data for whatever tests are made at the workbench, which has not been fully defined.

As illustrated in Figure 2-2 Tester's Workbench, the solid line encircling the workbench defines the scope of activities the worker performs at that workbench. Note that a single worker may have several workbenches, or several workers may use a single workbench. The input products are prepared at the previous workbench. The worker should validate that the input products contain the attributes needed for their workbench activities. If the input products fail to meet the entrance criteria, they should be rejected and reworked prior to starting the next workbench's activities.

The objective of the workbench is to produce the defined output products in a defect-free manner. The procedures and standards established for each workbench are designed to assist in this objective. If defect-free products are not produced they should be reworked until the defects are removed or, with management's concurrence, the defects can be noted and the defective products passed to the next workbench. This might be done because of scheduling constraints.

The worker performs defined procedures on the input products in order to produce the output products. The procedures are step-by-step instructions that the worker follows in performing his/her tasks. Note that if tools are to be used, they are incorporated into the procedures. It is important that tool usage becomes mandatory, not optional.

The standards are the measures that the worker uses to validate whether or not the products have been produced according to specifications. If they meet specifications, they are quality products or defect-free products. If they fail to meet specifications or standards they are defective and subject to rework.

It is the execution of the workbench that defines worker product quality, and is the basis for productivity improvement. Without process engineering, the worker has little direction or guidance to know the best way to produce a product, and to determine whether or not a quality product has been produced.

2.2.4 Responsibility for Building Work Processes

It is important that organizations clearly establish who is responsible for developing work processes (i.e., policies, procedures, and standards). Responsibility must be assigned and fulfilled. Having them developed by the wrong group can impede the effectiveness of the standards program.

2.2.4.1 Responsibility for Policy

IT management is responsible for issuing IT policy. Policies are the means which senior management uses to run their area. Again, policies define direction and by definition are general rules or principles. It is the standards, which will add the specificity needed for implementation of policies.

Policies define the intent of management. For example, IT project personnel need direction in determining how many defects are acceptable in their products. If there is no policy on defects each worker decides what level of defects is acceptable. However, if the organization issues a quality policy indicating that it is a policy to produce “defect-free products and services,” then the individual workers are provided the needed direction.

Policies are needed in all major areas in which the IT function conducts business, such as:

- Building Systems
- Testing Systems
- Maintaining Systems
- Operating Systems
- Quality of Systems
- Security of Systems
- Allocation of Resources
- Planning for Systems

- Training Personnel

These areas may actually result in multiple policies. For example, in software test planning there could be separate policies on scheduling, budgeting, risk assessment, and tool selection.

The key concepts that need to be understood on policies are:

- Policies are developed by senior management. (Note that in some instances subordinates develop the policy, but senior management approves it.)
- Policies set direction but do not define specific products or procedures.
- Policies are needed in areas that cause problems. For example, the transfer of developed applications from one group to another, or one person to another.
- Policies define the areas in which processes will be developed. (Note that if there are no policies, there should by definition be no standards or procedures in that area.)

2.2.4.2 Responsibility for Standards and Procedures

The workers who use the procedures and are required to comply with the standards should be responsible for the development of those standards and procedures. Management sets the direction and the workers define that direction. This division permits each to do what they are best qualified to do.

The workers generally know better than management what internal IT products are needed to meet a specific task objective, and how to produce those products. Failure to involve workers in the development of standards and procedures robs the company of the knowledge and contribution of the workers. In effect, it means that the people best qualified to do a task (i.e., development of standards and procedures) are not involved in that task. It does not mean that every worker develops his own procedures and standards, but that the workers have that responsibility and selected workers will perform the tasks.

IT management needs to define the hierarchical structure, which enables workers to develop standards and procedures. IT management must also encourage and motivate workers to fulfill that responsibility and then reward them for compliance.

The key concepts of a process engineering program are:

- Management provides an organizational structure for the workers to develop their own standards and procedures
- The program is driven by management policies
- Absolute compliance to standards and procedures is required
- A mechanism is provided for the continual maintenance of standards and procedures to make them more effective

Please note that he software tester should be the owners of test processes—and thus involved in the selection, development and improvement of test processes.

2.2.4.3 Test Process Selection

Selecting, developing, and acquiring work processes is an overall IT organization responsibility. Normally there is a function that performs this activity for the entire IT organization – frequently called a process engineering function. Software testers need to both understand how the activity operates AND participate when test processes, and related processes, are selected and put into practice.

An effective process engineering program does not happen without an active process engineering program. The program can be under the direction of the quality assurance group, a Process Engineering Committee, a Process Engineering Manager, or the direction of senior IT management. However, without a "catalyst" to push and prod for processes, the program frequently flounders.

An active program begins with a self-assessment, continues with the formation of the proper organization, and continues as an ongoing active program as long as products are made and modified.

The process engineering program should be proactive, rather than a reactive program. It should take an active role in determining areas in which standards and policies are needed, and then taking the steps necessary to ensure they are developed. Reactive programs do not provide the same benefits to an organization as a proactive program.

IT groups should develop a plan for implementing and operating a process engineering program. This would require a policy for standards, and a charter or job description for the function. These need to be customized for each organization. The specific components that need to be addressed include:

- Building a Process Engineering Organization
- Developing a Standard and Procedure for Standards
- Planning for Standards
- Writing, Storing, and Retrieving Standards and Procedures
- Enforcing Standards

2.2.4.4 Building a Process Engineering Organization

The structure that is put in place to develop and update policies, standards, and procedures must involve both staff and management. The process engineering program should be directed by management, but implemented by staff. This is the same basic approach used for any other activity undertaken in the IT department.

The implementation of processes by management (i.e., senior management and/or management staff functions such as quality assurance) is wrong. It makes no more sense to have management write processes, than it does to have management design systems and write software. As stated previously, the individual best equipped to perform that job should perform the work.

Some guidelines on establishing an organizational structure for process engineering are:

- Establish a Process Engineering Committee compromised of the most senior IT managers possible.
- Represent all IT organizational areas on the Process Engineering Committee.
- Appoint an individual as the Process Engineering Manager. (Note that this can be a part-time assignment.)
- Appoint Ad Hoc Committees (i.e., special task forces) to develop individual standards and procedures.
- Let the Standards Ad Hoc Committees develop the technical standard.

2.2.4.4.1 Recommended Standards Organizational Structure

The recommended organizational structure is comprised of the following three components:

- Process Engineering Manager

  A full-time or part-time individual responsible for managing the standards program.

- Process Engineering Committee

  A policy-setting board, which runs the standards program.

- Ad Hoc Committee

  Small groups which develop single standards at a time.

A suggested organizational chart for process engineering is illustrated in Figure 2-1. This shows both the Process Engineering Manager and a Process Engineering Committee as a staff function to the most senior IT Manager. The IT Manager is defined as the top operational officer in IT, the one to whom systems and programming, database, data communications, and operations, reports. The Process Engineering Committee should be comprised of the IT Manager and all of the managers who directly report to that individual, or an acceptable substitute. An acceptable substitute is one who can speak for the manager, and feels comfortable about it. If the Process Engineering Committee is comprised of lower-level personnel it lacks the necessary clout and image needed to promote processes effectively in the organization. Note that in large organizations, lower-level managers may be selected to represent more senior managers.

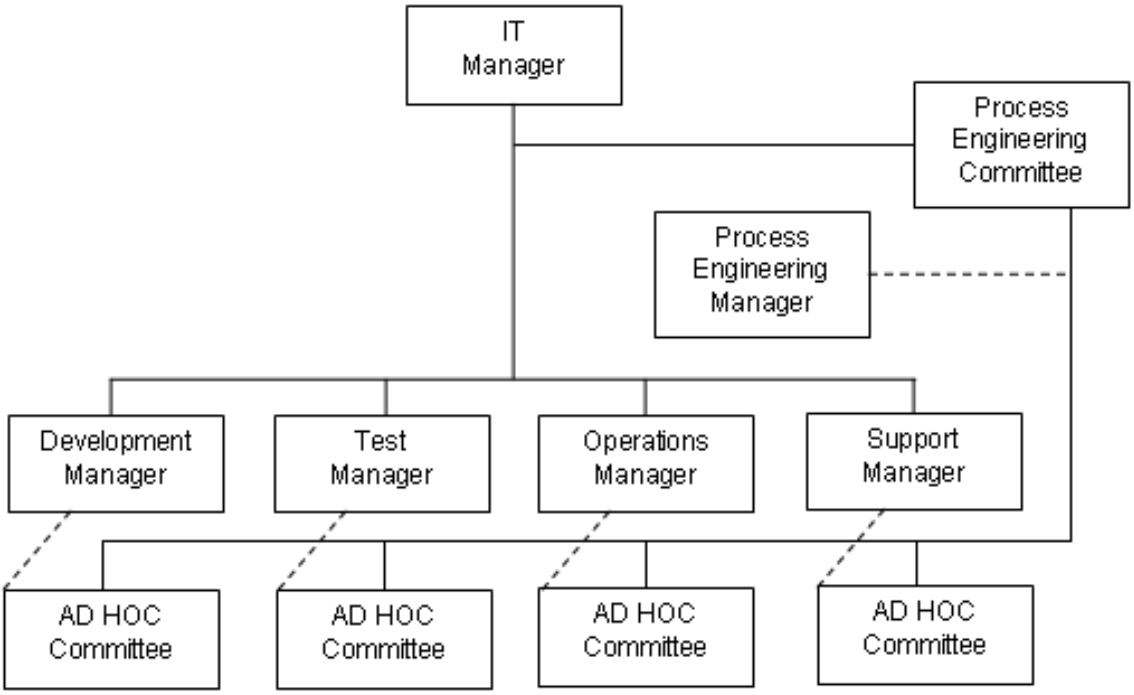


Figure 2-1 Recommended Standards Organizational Chart

The Ad Hoc Committees are comprised of the individuals responsible for developing the procedures and the standards. Their peers should respect the individuals selected on the Ad Hoc Committees. It is the Ad Hoc Committees who develop the standards, and then have a moral obligation to ensure that their peers follow them.

2.2.4.4.2 Role of Process Engineering Manager

The responsibilities of the Process Engineering Manager include:

- Promote the concept of process engineering.

  The Process Engineering Manager must actively develop and implement programs that create enthusiasm and motivation for the program. This includes development of processes, recommendations for improvement of processes, and personal enforcement of processes.

- Be the driving force behind processes.

  The Process Engineering Manager must be the leader of process engineering and the advocate for processes. The manager must be the one that prods and encourages the Process Engineering Committee, Ad Hoc Committees, and involved parties to do their job.

- Administer the standards program defined by the Process Engineering Committee.

  The Process Engineering Manager must make the program work through personal energy, enthusiasm, and hard work.

- Be a resource to the Process Engineering Committee and Ad Hoc Committees.

  The Process Engineering Manager should be available to these committees to ensure that both organizations are fulfilling their assigned responsibilities.

- Ensure involved parties are adequately trained.

  The Process Engineering Manager must ensure that the Process Engineering Committee and Ad Hoc Committees are trained in how to fulfill their function, and that the users of the standards and procedures are trained in how to use them. This does not mean that the Process Engineering Manager per se should run the training sessions but, rather, must ensure they are run.

2.2.4.4.3 Role of the Process Engineering Committee

The role of the Process Engineering Committee is to:

- Accept topics for processes.

  The members of the Process Engineering Committee, including the Process Engineering Manager, can nominate activities for processes. These activities should be related to the policies of the department. The Process Engineering Committee votes to determine whether an activity will be accepted or rejected as a process activity.

- Set priority for implementation of processes.

  The Process Engineering Committee should determine the sequence in which standards are to be implemented. Ideally, this includes the approximate date when the standards should be developed and become effective.

- Obtain the resources necessary to develop the process.

  Someone on the Process Engineering Committee should accept responsibility for forming an Ad Hoc Committee. This means finding a chairperson (i.e., sponsor for the committee) and ensuring that the appropriate resources are available for that Ad Hoc Committee to develop the process.

- Approve or reject developed processes.

  The Ad Hoc Committee should send the standards to the Process Engineering Committee for approval. The committee approves as is, approves subject to modification, or rejects the standard.

Any member of the Process Engineering Committee should not spend more than approximately one hour per month in a Process Engineering Committee meeting. However, between meetings the individual members would be disseminating approved standards to their function, obtaining resources to develop new processes, and working with or providing support to the Ad Hoc Committees.

2.2.4.4.4 Role of the Ad Hoc Committee

The responsibilities of the Ad Hoc Committee are to:

- Gain representatives from all involved areas.

Each activity within the organization that is affected by the process should have a representative on the Process Engineering Committee. Note that in some cases, one representative may represent more than one area.

- Ensure that the committee has between three and eight members in size.

  However, if the process is somewhat perfunctory, such as assigning a job number, it may be accomplishable by a single individual.

- Create the standard and procedure.

  This means that the final written standard and procedure does not have to be written by the Ad Hoc Committee, just the material needed to write the final standard and procedure.

- Coordinate reviews of the standard with involved parties.

  The Ad Hoc Committee has the obligation to ensure that all involved parties have had an opportunity to react and comment on the standard and procedure. Hopefully, this task would also obtain concurrence from the involved parties. The work of the Ad Hoc Committee continues until consensus is developed. Note that in some instances, if consensus is impossible, IT management must make a decision regarding direction.

- Periodically review and update the standards and procedures previously developed by the Ad Hoc Committee.

  Approximately annually, the Ad Hoc Committee should reconvene to quickly review the standards and procedures to see that they are still current and applicable. If not, the Ad Hoc Committee should make any necessary changes.

2.2.4.4.5 Selecting Process Engineering Committee Members

The general guideline for selecting members of the Process Engineering Committee is to aim high. Having members of senior IT management on the Process Engineering Committee solves many problems. First, it draws attention to the importance of the committee through committee assignments. Second, it solves the problem of people not attending the meeting, because when senior managers attend, everyone involved attends. Third, it places the responsibility for quality clearly on the shoulders of the individuals who should have that responsibility – management.

The makeup of the Process Engineering Committee is important. Some guidelines in selecting members are:

- Select the highest-level manager who will accept the position
- Assign individuals who are supportive and enthusiastic over standards
- Make long-term assignments to the Process Engineering Committee
- Select individuals who are respected by their peers and subordinates
- Ensure the appropriate areas of interest are involved

Some of the above criteria may be mutually exclusive. For example, a senior manager may not be enthusiastic over standards. On the other hand, assignment to the committee and involvement in standards may create that enthusiasm. Go for the high-level managers first.

Some guidelines to ensure the appropriate areas of interest are included in developing standards are:

- Is every organizational function within IT involved?
- Are the activities that interface to the IT function involved; for example, key users and customers?
- Are activities having a vested interest involved, such as auditors?
- Are all the IT "businesses" represented such as the project leaders, systems programmers, data library, security, help desk, and so forth?

The technical and functional challenges of each activity should be identified. For example, if the help desk is an activity, what technical and functional challenges do the help desk have in fulfilling its mission.

Candidates should be selected who can adequately represent the activity, and are aware and knowledgeable in the technical and functional challenges. The candidates should be listed in order of desirability. The manager of the Standards Committee, or the Standards Manager, should then visit and make an appeal to those individuals for participation on the committee. Note that clearly defining the technical and functional challenges, and the IT activity, helps get the right people on the committee. Individuals are more willing to serve when they know they are on for a specific purpose. Whether they accept or reject can then be indicated on the worksheet.

2.2.4.5 Professional Test Standards

Professionals in any industry need to be aware of industry standards that apply to their domain. Several organizations publish standards covering the test process, activities, and work products. Test professionals should be familiar with the standards published by organizations such as the International Standards Organization (ISO), U.S. Department of Commerce (National Institute of Standards and Technology), and IEEE. In addition, testers should know when these standards apply, and how to obtain the latest versions.

2.2.4.5.1 IEEE Software Engineering Standards

IEEE publishes a body of standards that address software engineering and maintenance. A portion of these standards addresses software testing and test reporting. Listed below are the current standards that apply to the test process.

829-1998IEEE Standard for Software Test Documentation

830-1998IEEE Recommended Practice for Software Requirements Specifications

1008-1987 (R1993)IEEE Standard for Software Unit Testing (ANSI)

1012-1998 IEEE Standard for Software Verification and Validation

1012a-1998IEEE Standard for Software Verification and Validation – Supplement to 1012-1998 – Content Map to IEEE 12207.1

1028-1997IEEE Standard for Software Reviews

Other professional organizations have similar standards. The CSTE candidate should have an overall familiarity with the purpose and contents of these standards.

2.2.5 Analysis and Improvement of the Test Process

Studies at many IT organizations have indicated that testers make more defects in performing test activities than developers do in performing developmental activities. For example, if the developers make three defects per function point of logic; testers would make more than three defects in testing that function point of logic.

There are two reasons for this high defect rate. The first is that test processes are less mature than most developmental processes. In this case, mature means the variability in performing the activity. Generally the more variability in a test process the higher the defect rates. The second reason is that testers do not have a quality control process over their test process.

Figure 2-2 illustrates the two test processes. One process performs testing and one parallel process checks the performance of testing.

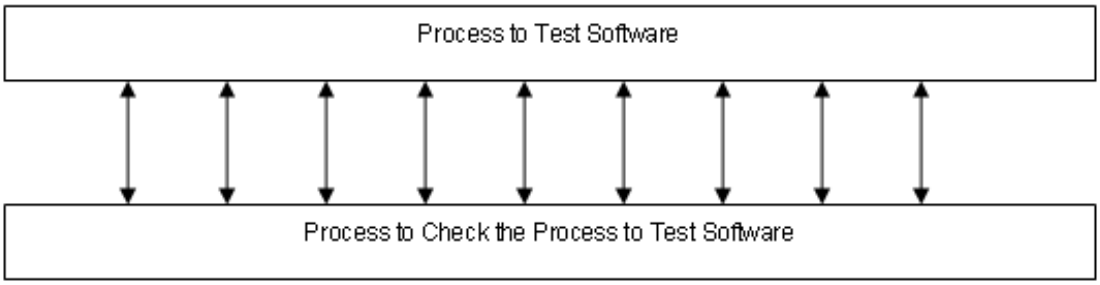


Figure 2-2 The Two Testing Processes

Let's look at an example. If testers were to develop test transactions to test transactions a, b, and c, there would be a quality control process that checks to assure that those three test transactions were actually prepared, and the transactions contained necessary data to perform the specified testing. Quality control can be performed by the individual who does the work, or by another individual. For example, testers may be involved in performing a code inspection process on programming code prior to unit testing. This same inspection process can be used to inspect the test plan, test data and the documentation produced from testing. Other testers would perform the test inspection process on the work performed by another tester.

Testers are the quality control process for developers. They are checking to see that the work done by developers meets the specifications for those developer-produced products. The same concepts need to be applied to the test process. If high-quality work is needed, quality control should be an integral part of the test process. Sometimes quality control can be automated, other times it must be performed manually.

2.2.5.1 Test Process Analysis

Test quality control is performed during the execution of the process. In other words, a test task is performed and then the quality of that test task is checked. Analysis is performed after the test process is completed. This analysis can be performed by the test group, or it can be performed by another group within the IT organization such as the quality assurance function or process engineering activity.

Test analysis can only be performed if adequate data is maintained during the test process. A determination needs to be made as to the type of analysis that will be performed, and then that data collected for analysis purposes. Ideally, the data that should be collected is data that is produced as a by-product of testing. This data tends to be more reliable than data which is collected manually during and after the execution of a test activity.

If data is to be collected during the test process for analysis purposes, the process must include the capabilities to collect that data. The type of data collected might include time spent on a specific activity, rework, defects caused by the tester or the test process, surveys of testers and stakeholders in the test process, inquiries by developers and/or users regarding test activities, test reports and test results, as well as logs of the specific activities performed by testers.

There are many reasons why a test group may want to perform an analysis of the test process. The following analyses are the most common in the test industry. Each is discussed in more detail below.

- Effectiveness and efficiency of test processes
- The test objectives are applicable, reasonable, adequate, feasible, and affordable
- The test program meets the test objectives
- The correct test program is being applied to the project
- The test methodology is used correctly
- The task work products are adequate to meet the test objectives
- Analysis of the results of testing to determine the adequacy of testing

2.2.5.1.1 Effectiveness and Efficiency of Test Processes

Estimates show that in many organizations testing consumes 50% of the total developmental cost. In fact, this often has been referred to as “test and fix.” Many organizations are not aware of the division between doing work, testing work, and rework because all of those activities are charged to a project without dividing the charge among the three previously described activities.

Effectiveness means that the testers completed their assigned responsibilities. As previously stated, this should be completion of the activities included in the test plan. Efficiency is the amount of resources and time required to complete test responsibilities.

There is normally a trade-off between efficiency and effectiveness. Since there are usually time and budget constraints the testers should be looking for test processes which maximize the two variables of effectiveness and efficiency.

2.2.5.1.2 The test objectives are applicable, reasonable, adequate, feasible, and affordable

The test objectives are the responsibilities assigned to testers to test an individual software system. The test objectives should be incorporated into the test plan. The test plan will then define how those test objectives will be accomplished during the execution of testing.

Test objectives may be assigned to testers which are not applicable, not reasonable, not adequate, not feasible and not affordable. The reasons test objectives may not achieve these objectives include:

- Testers do not have the needed competencies to meet the test objectives
- Test tools are not available which can complete the test objectives within reasonable time or resource constraints.
- Test objectives are not objectives that should be assigned to testers, such as whether the software is appropriately aligned to the corporate business objectives.
- The test objectives overlook what would reasonably be incorporated into software testing, such as identifying implemented functionality not requested by the user of the software.

2.2.5.1.3 The test program meets the test objectives

Surveys by the Quality Assurance Institute have indicated that only approximately one half of the software testing groups develop test plans. Of those groups that develop test plans, approximately 50% do not follow those plans during the test execution activities.

Failure to plan, or follow the plan limits the ability of the software testers to determine whether or not the written or implied test objectives have been achieved. Very few testers can prepare reports at the end of testing indicating whether or not the stated test objectives have been implemented.

This problem is frequently compounded by the fact that the software specifications have changed from the time the test objectives were defined. This may mean that the objectives need to be modified or changed based on the change in specification. However, if the test plan and test objectives are not changed to reflect those different specifications, whether the original objectives are met or not met may be immaterial based on the new specifications.

2.2.5.1.4 The correct test program is being applied to the project

The test environment specifies the approach and test processes that should be used to test software. The environment should also specify how those general test processes should be

customized based on different testing needs and different software development methodologies used.

Depending upon management support for using and following the appropriate test processes, the testers may or may not follow those processes. In some IT organizations more emphasis is placed on meeting implementation schedules than on following the test processes.

If the correct test processes are not followed improvement of those processes is severely limited. In other words, if specific processes or tasks within processes are not followed and the results are undesirable, it may be difficult to determine whether the cause is the process, or the changes to the process made by a single software test team.

There are two aspects of determining whether the correct test program was applied:

- Was it applied as developed and incorporated into the test environment?
- Was it corrected to adjust for changing test conditions and different software development methodologies?

2.2.5.1.5 The test methodology is used correctly

This analysis requires an effective quality control process. Without quality control in place it may be difficult and expensive to identify the root cause of a test failure. For example, if a tester selects an alternate tool rather than the specified tool, but no record is maintained of what tool is actually used, then compliance to the test methodology is difficult to determine.

If adequate quality control records are maintained over testing it is possible to identify where the methodology was not complied with. Non-compliance is not necessarily bad, because the process may not be doable as developed. In other words if the tester followed the process exactly as specified an undesirable result would occur. Testers frequently know this and make changes during test execution so that their test objectives can be met. However, for the processes to be continuously improved they must know:

- Whether or not the process was complied with
- Whether the tasks not adequately defined can be followed or if followed will not work

2.2.5.1.6 The task work products are adequate to meet the test objectives

During development and execution of testing, testers produce many different test products. Among these are risk analysis reports, the test plan, test data specifications, test matrices, test results, test status reports and final test reports.

The following two analyses need to be performed on these work products:

- Are they adequate to ensure that the test objectives can be met?
- Are they products, or parts of test products that are not needed to meet the test objectives?

The result of this analysis is to ensure that the appropriate information is prepared and available for the testers and other stakeholders to use. Reports that contain unwanted or unnecessary information make the task of using the work products more difficult.

2.2.5.1.7 Analysis of the results of testing to determine the adequacy of testing

Two processes must be in place in order to manage testing and assess the adequacy of testing. These are a monitoring process which monitors progress and a communication process which provides information to both monitor and act on monitoring information. Skill Category 6, the Test Reporting Process, provides the monitoring and communication processes.

Many test organizations develop "tester's dashboards" comprised of key indicators to facilitate this monitoring process. The key indicators on the dashboard are those measurements which are needed to assure that testing is performed in an effective, efficient and timely manner. Some of the frequently used key indicators are:

- Budget status
- Schedule status
- Requirements tested correctly
- Requirements tested but not correct
- Severity of recorded defects
- Age of recorded but not corrected defects
- Percentage of test plan completed
- Stakeholders satisfaction

To determine the adequacy of the test process, adequacy must be defined quantitatively. For example adequacy might define that checkpoints are completed no later than two days following the checkpoint date. If that standard/goal is met then the test program is deemed to be adequate. Another adequate test standard/goal might be that all high-priority and medium-priority requirements are tested, and at least 80% of the low-priority requirements are tested by implementation date.

2.2.5.1.8 Adequate, not excessive, testing is performed

Implementing defective software is a risk, testing is a control designed to eliminate or minimize that risk. However, the cost of controls should never exceed the maximum potential loss associated with risk. For example if the cost of testing requirement X is $10,000 and the potential loss if requirement X does not work correctly is $500, that testing would not be warranted. Perhaps a very minimal test that would cost $200 that could reduce the risk from $500 to $100 would be economically feasible.

Adequate testing needs to be defined to ensure that adequate, but not excessive, testing is performed. To do this testing must establish guidelines as to what is excessive testing. These guidelines might include:

- A potential test objective is not wanted by the stakeholders, for example testing the efficiency of the software when operating on the hardware
- The cost of testing exceeds the potential loss of the risks associated with not testing
- The cost of acquiring and using automated tools exceeds the costs of performing the same test manually

- Testing is assigned to an individual who does not have the appropriate competencies to perform the test, and thus would be very inefficient in performing the test

To ensure that testing is not excessive, testers must continually question the value of each test they perform. They must also look at the magnitude of the risks that are associated with the software being tested. Once the magnitude of the risks is known, the testers can focus their limited resources on the high-risk attributes associated with the software.

2.2.5.2 Continuous Improvement

Process improvement is best considered as a continuous process, where the organization moves continually around an improvement cycle. Within this cycle, improvement is accomplished in a series of steps or specific actions. An important step in the improvement cycle is the execution of data gathering to establish the initial state, and subsequently to confirm the improvements. A testing process assessment is one excellent way to determine the status of your current test process.

Assessment is the thoughtful analysis of testing results, and then taking corrective action on the identified weaknesses.

Testing process assessment is a means of capturing information describing the current capability of an organization's test process. Assessment is an activity that is performed either during an improvement initiative or as part of a capability determination. In either case, the formal entry to the process occurs with the compilation of the assessment input, which defines the:

- Purpose – why it is being carried out
- Scope – which processes are being assessed and what constraints (if any) apply
- Any additional information that needs to be gathered
- The input also defines the responsibility for carrying out the assessment

Process assessment is undertaken to measure an organization's current processes. An assessment may be conducted as a self-assessment, an assisted self-assessment, or an independent assessment. The assessment may be discrete or continuous. A team with, or without, tool support typically carries out a discrete assessment manually. A continuous assessment may use manual methods or automated tools for a data collection. Whatever form of assessment is used, a qualified assessor who has the competencies required oversees the assessment.

An assessment is carried out by assessing selected processes against a compatible model of good engineering practice created from, or mapped to, the defined reference model. The reference model defines processes characterized by statements of purpose, and attributes that

apply across all processes. The process attributes are grouped into process capability levels that define an ordinal scale of capability.

The assessment output consists of a set of attribute ratings for each process instance assessed, and may also include the capability level achieved. Process assessment is applicable in the following circumstances:

- Understanding the state of processes for improvement
- Determining the suitability of processes for a particular requirement or class of requirements
- Determining the suitability of another organization's processes for a particular contract or class of contracts.

The framework for process assessment:

- Encourages self-assessment
- Takes into account the context in which the assessed process operates
- Produces a set of process ratings (a process profile) rather than a pass or fail result
- Addresses the adequacy of the management of the assessed processes through generic practices
- Is appropriate across all application categories and sizes of organization

The sophistication and complexity required of a process is dependent upon its context. This influences how a qualified assessor judges a practice when assessing its adequacy, and influences the degree of comparability between process profiles.

Within a process improvement context, assessment provides the means of characterizing the current practice within an organizational unit in terms of capability. Analysis of the results in the light of business needs identifies strengths, weaknesses, and risks inherent in the processes. This, in turn, leads to the ability to determine whether the processes are effective in achieving their goals and to identify significant causes of poor quality or overruns in time or cost. These provide the drivers for prioritizing improvements to processes.

One of the most commonly identified weaknesses in software testing has been the lack of facts (metrics), and without facts there is no reason to take action (improvement). Once appropriate measures are identified, tracked, and analyzed, then a plan for continuous improvement can be implemented.

It is important to understand that the concept of "measurement first and action second", is most effective when the measures are very specific. Measurement must be able to determine the effect of the actions. For example, the metric approach fulfills this requirement in that it shows very specific relationships. Using this concept, if action is taken by changing one of the metric variables, the result of that action can be quickly measured.

Changing the variable in one metric can usually be measured by the changes to another metric. As another example, if the number of defects detected after the system goes into production is higher than expected or desirable, then appropriate and timely action should be taken. That action might be to increase the number of test steps. Obviously, this will increase

testing costs with the objective of improving the quality of the product by reducing defects. If, after analysis, it is demonstrated that increasing the emphasis on test steps has had the desirable effect, then these additional steps should be incorporated into the normal testing process. If the opposite is true, then the action did not produce the desired effect and the time and resources spent were less than effective and the actions should be discontinued and another attempted. The process continues until the appropriate and effective improvement mechanisms are uncovered and included in the normal process.

2.2.5.3 Test Process Improvement Model

A model for test process improvement has these eight steps:

1. Examine the Organization's Needs and Business Goals

   A process improvement program starts with the recognition of the organization's needs and business goals, usually based on the main drivers and stimuli identified. From an analysis of the organizations needs and existing stimuli for improvement, the objectives of process improvement are identified and described. The final stage of the preliminary definition of the goals for the improvement program is setting the priorities of the process improvement objectives.

   Once the analysis of the organization's needs and business goals has been completed, it is essential to build executive awareness of the necessity for a process improvement program. This requires both managerial and financial commitments.

   The objectives of such a process improvement program should be clearly stated and understood, and expressed using measurable process goals. The process improvement program should form part of the organizations overall strategic business plan.

2. Conduct Assessment

   The assessment should be conducted according to a documented process. Assessors must have access to appropriate guidance on how to conduct the assessment and the necessary competence to use the tools.

   Each process is assessed by detailed examination. A rating is assigned and validated for each process attribute assessed. In order to provide the basis for repeatability across assessments, the defined set of indicators is used during the assessment to support the assessors' judgment in rating process attributes.

   Objective evidence based on the indicators that support the assessors' judgment of the ratings are recorded and maintained to provide the basis for verification of the ratings.

3. Initiate Process Improvement

   The process improvement program is a project in its own right, and planned and managed accordingly. A plan should be produced at the beginning of the program and subsequently used to monitor progress. The plan should include the relevant background, history, and the current status, if possible expressed in specific, numerical terms. The input derived from the organization's needs and business goals provide the main requirements for the plan.

   The plan should include a preliminary identification of the scope in terms of the boundaries for the program and the processes to be improved. The plan should cover all

the process improvement steps, although initially it may give only outline indications of the later stages. It is important to ensure that key roles are clearly identified; adequate resources allocated, appropriate milestones and review points established, and all risks are identified and documented in the plan. The plan should also include activities to keep all those affected by the improvement informed of progress.

4. Analyze Assessment Output and Derive Action Plan

   Information collected during the assessment, in particular the capability level ratings, the generic practice ratings, and the base practice ratings, is first analyzed, and a plan of action is derived. This consists of the following activities:

   - Identify and prioritize improvement areas.
   - Analyze assessment results.

     Analysis of the results provides information about the variability as well as current strengths and weaknesses and indicates opportunities for improvement.
   - Analyze the organization's needs and improvement goals.

     The processes and their relationships are analyzed to evaluate which have direct impact on the goals identified. A priority list of processes to be improved is then derived.
   - Analyze effectiveness measurements.
   - Analyze the risks in not achieving improvement goals.

     The impact of failing to achieve improvement goals is evaluated in order to understand the urgency and to set the priority of initiatives.
   - Analyze risks of improvement action failure.

     The potential risks of failure of an improvement action is analyzed to support the definition of priorities and to assure commitment and organizational support.
   - List improvement areas.

     A prioritized list of improvement areas is provided as a result of analyzing all the factors listed above.
   - Define specific improvement goals and set targets.

     Targets for improvement should be quantified for each priority area.
   - Derive action plan.

     A set of actions to improve processes should be developed. Care should be taken to select a set of actions, which support each other in achieving the complete set of goals and targets. It is desirable also to include some improvement actions, which yield clear short-term benefits in order to encourage acceptance of the process improvement program.

5. Implement Improvements

   A process improvement action plan is implemented in order to improve the process. Implementation may be simple or complex depending on the contents of the action plan and the characteristics of the organization.

   In practice, several process improvement projects will be initiated, each concerned with implementing one or more process improvement actions. Such projects will often not

cover only initial implementation of improvements. Four main tasks are involved in each process improvement project:

- Operational approach to implementation.

  Where there are alternative operational approaches to implementation, they should be evaluated and the most suitable selected. It may be possible to implement in small steps through piloting in a selected unit or throughout the whole organization at the same time, or somewhere between these two extremes. Among the factors to consider are costs, time scales, and risks.

- Detailed implementation planning.

  A detailed implementation plan is then developed. The process improvement project may need to carry out a deeper analysis of improvement opportunities than that already carried out. Those implementing the actions and those affected by them should be involved, or be consulted, in developing the plan and in evaluating alternatives, in order to draw both on their expertise and enlist their cooperation.

- Implementing improvement actions.

  During this activity, it is critical for successful improvement that due account is taken of human and cultural factors.

- Monitoring the process improvement project.

  The organization's management against the process improvement project plan should monitor the process improvement project. Records should be kept for use to both confirm the improvements, and to improve the process of process improvement.

6. Confirm Improvements

   Management as well as stakeholders must be involved both to approve the results and to evaluate whether the organization's needs have been met. If, after improvement actions have been taken, measurements show that process goals and improvement targets have not been achieved, it may be desirable to redefine the project or activity by returning to an appropriate earlier step.

   - Improvement targets.

     Current measurements of process effectiveness should be used to confirm achievement of process effectiveness targets. The possibility of having introduced desirable or undesirable side effects should be investigated.

   - Confirm achievement of targets.

     A further process assessment should be used to confirm achievement of targets expressed as process capability levels. Where several improvement projects were undertaken, however, consideration should be given to a reassessment of wider scope to check for potential side effects arising from the parallel improvement actions.

   - Organizational culture.

     The effect of the improvements on organizational culture should be reviewed to establish that desired changes have taken place without undesirable side effects.

   - Re-evaluate risks.

The organization should re-evaluate the risks of using the improved process to confirm that they remain acceptable, and if they are not, determine what further actions are required.

- Re-evaluate cost benefit.

  The costs and benefits of the improvements may be re-evaluated and compared with earlier estimates made. These results are useful to support planning of subsequent improvement actions.

7. Sustain Improvement Gains

   After improvement has been confirmed, the process needs to be sustained at the new level of performance. This requires management to monitor institutionalization of the improved process and to give encouragement when necessary. Responsibilities for monitoring should be defined, as well as how this will be done by using appropriate effectiveness measurements.

   If an improved process has been piloted in a restricted area or on a specific project or group of projects, it should be deployed across all areas or projects in the organization where it is applicable. This deployment should be properly planned and the necessary resources assigned to it. The plan should be documented as part of the process improvement project plan or the process improvement program plan as appropriate.

8. Monitor Performance

   The performance of the process should be continuously monitored. New process improvement projects should be selected and implemented as part of a continuing process improvement program, since additional improvements are always possible.

   - Monitoring performance of the process.

     The performance of the process should be monitored as it evolves over time. The effectiveness and conformance measures used for this should be chosen to suit the organization's needs and business goals, and should be regularly reviewed for continuing suitability. The risks to the organization and its products from using the process should also be monitored and action taken as risks materialize or become unacceptable.

   - Reviewing the process improvement program.

     Management should review the process improvement program regularly. Further process assessments can be an important component of the continuing improvement program. The extent to which improved processes have been institutionalized should be considered before scheduling further process assessments. It may be more cost-effective to delay assessing a process until improvements have been fully deployed, rather than expend resources assessing a process, which is in transition, when the results can be difficult to interpret.

The bottom line of assessment is making application system testing more effective. This is performed by a careful analysis of the results of testing, and then taking action to correct identified weaknesses. Facts precede action, and testing in many organizations has suffered from the lack of facts. Once those facts have been determined, action should be taken.

The “measurement first, action second” concept is effective when the measurement process is specific. Measurement must be able to determine the effect of action. For example, the metric approach fulfills this requirement in that it shows very specific relationships. Using this concept, if action is taken by changing one of the metric variables, the result of that action can be quickly measured.

Changing the variable in one metric can normally be measured by the change in another metric. For example, if the number of defects detected after the system goes operational is higher than desirable, then action should be taken. The action taken might be to increase the number of instructions exercised during testing. Obviously, this increases test cost with the hopeful objective of reducing undetected defects prior to operation. On the other hand, if increasing the number of instructions executed does not reduce the number of defects undetected prior to production, then those resources have not been used effectively and that action should be eliminated and another action tried.

Using the measurement/action approach, the variables can be manipulated until the desired result is achieved. Without the measurement, management can never be sure that intuitive or judgmental actions are effective. The measurement/action approach works and should be followed to improve the test process.

2.2.5.4 Test Process Alignment

In establishing the test environment management must assure that the mission/goals of the test function are aligned to the mission/goals of the organization. For example if a goal of the organization is to have high customer satisfaction, then the mission/goal of the test function would be to do those activities which lead to high customer satisfaction of the testers customers. This may mean that they focus testing more on what the customer needs, as opposed to the defined specifications.

Figure 2-3 is an example of how that alignment occurs. This figure is a test process alignment map. The objective of the map is to assure that the test processes are aligned with the organizational user testing goals. One axis of the matrix lists the organizational user testing goals, and the other axis the test processes. A determination is then made as to whether or not a specific test process contributes to an organizational and/or user testing goal. In this example a check-mark is put in the intersection indicating which test process contributes to which goal.

	Test Processes				
Organizational and User Testing Goals	**Risk Analysis**	**Test Planning**	**Test Data Preparation**	**Test Reporting**	**Measurement of Results**
Software meets requirements		√	√		√
Software meets user needs	√	√	√		√
Software easy to use		√	√		√
Five sigma defects in operational software	√	√		√	√

Figure 2-3 Example of a Test Process Alignment Map

The assessment of this map is two-fold:

1. To determine that there are adequate processes in place to achieve the goal
2. To determine that there are no goals without processes or processes that do not contribute to defined goals

2.2.5.5 Adapting the Test Process to Different Software Development Methodologies

In the initial days of software development, testing was considered a phase of development. The testing phase began after software was developed. However, that proved to be a very costly approach as the cost of correcting defects rose significantly as the development process proceeded. Many have estimated that it costs ten times as much to correct a defect found in a test phase, than if that same defect had been found during the requirements phase.

When testing is viewed as a life cycle activity, it becomes an integral part of the development process. In other words as development occurs testing occurs in conjunction with development. For example, when requirements are developed, the testers can perform a requirements review to help evaluate the completeness and correctness of requirements. Note that testers may be supplemented with subject matter experts in some of these tests, such as including users in a requirements review.

Understanding the software development process used in his/her organization is important in conducting software testing for the following three reasons:

1. Understanding the developmental timetable and deliverables.

   If testers are going to perform test activities throughout the development process they need to know what developmental products will occur at what time. For example, if testers

plan on performing a requirements review, they need to understand the requirements document, and when those documents will be available for testing

2. Integrating the software testing process with the development process

   The test process is in fact a mirror image of the development process. Later in this category you will review an eleven-step software testing process that is related to the development process. Until the development process is known, the test process cannot be appropriately created and integrated into the development process. This integration is important so that adequate time and resources are allocated for both building the software and testing the software.

3. Understanding how software is developed

   It is difficult to effectively test the product if you don't know how the product was developed. It is for this reason that we differentiate white-box testing from black- box testing. If the tester is going to effectively perform white-box testing they need to understand the process by which the software products were developed. If they do not understand the process by which software is developed, they become forced to emphasize black-box testing.

There are literally thousands of software development and testing processes. A study a number of years ago indicated that in the United States alone there were 50,000 different software applications for calculating payroll. If a common business activity such as payroll can be performed using 50,000 different variations, there is no reason to suspect that there aren't 50,000 variations for building and testing software.

While there may be literally thousands of software development methodologies, the book "Quality Software Project Management" published by the Software Quality Institute, describes the following six major categories of developmental methodologies and then indicates the advantages and disadvantages of each.

- **Waterfall** - The waterfall methodology is the oldest software development methodology. It follows a logical progression of defining requirements, designing the system, building the system, testing the system, and placing the system in operation. It is primarily used for the development of systems that do batch processing. The waterfall methodology assumes that at the end of the requirements phase the requirements for development are known.
- **Prototype** - The prototype methodology assumes that the user does not have a rigid definition of requirements. Prototyping applies an "I'll know it when I see it" environment and produces an operational methodology of the user's requirements currently defined. The user can then see the system in operation and make changes for another generation or prototype of the system. The development of prototype versions continues until the user believes the system contains the requirements desired by the user. Used alone, the prototype methodology lacks the necessary controls to ensure correct and high-quality processing.
- **Rapid Application Development (RAD)** - The objective of the RAD methodology is a quick turnaround time with emphasis on requirements definition, which is accomplished by heavy user involvement throughout all developmental phases. RAD

makes extensive use of developmental tools. Much of its effectiveness and reduced cycle time is due to rapid movement from one phase of development to the next.

- **Spiral** - The spiral methodology focuses on objectives, alternatives, constraints, and risks. The spiral methodology, like RAD, heavily involves the user. At each part of the spiral, development, the risks, constraints, and alternative methods are evaluated. The objective is to use the best possible system from a business perspective, although development may require a longer developmental cycle than RAD.
- **Incremental** - The objective of the incremental methodology is to develop the system in parts (or increments). It is effective when single parts of the system can be used immediately without the full system being developed or when a particular part is critical to the success of the overall system, but there is some uncertainty as to whether that part can be effectively developed. The incremental method can be effective in reducing the risk of investing and building an entire system when the outcome is questionable.
- **The V Methodology** - The V methodology is mostly associated with software testing. It develops two processes: one for building the system and one for testing the system. The two processes are then interrelated as a V methodology. The V shows development on one side and testing on the other side. For example, during the requirements stage of development, the software acceptance testing side is developed. The V methodology assumes that approximately one-half of the total development effort will be spent on testing. The V then integrates testing so that testing is more effective and defects are uncovered earlier in the developmental process.

Testers testing software developed by a specific software development methodology need to:

1. Understand the methodology
2. Understand the deliverables produced when using that methodology
3. Identify compatible and incompatible test activities associated with the developmental methodology
4. Customize the software test methodology to effectively test the software based on the specific developmental methodology used to build the software

In customizing the software testing process to the different methodologies the tester needs to know:

- Will the users be involved in the developmental process?
- Do the users understand this developmental process?
- Are the user’s experts in their business activity?
- Is this project an enhancement to an existing software system developed using another methodology?
- Is high reliability an important aspect of this software development project and does the selected developmental methodology assure high reliability?
- Is the amount of changes expected to be implemented in the software system consistent with the capabilities of developmental methodology?

2.3 Test Tools

It is difficult to perform testing economically without the aid of automated tools. For example regression testing without a capture-playback tool would be very difficult to achieve manually. However, it is important that tools are selected to support the test methodology and thus their use should be mandatory and not optional.

The most important aspect of software testing tools is the process used to acquire those tools. Most of this component of the environment will focus on acquiring or developing testing tools.

2.3.1 Tool Development and Acquisition

The procedures developed for testers to follow during testing should include testing tools and techniques. The testing organization should select which testing tools and techniques they want used in testing, and then incorporate their use into testing procedures. Thus, tool and technique usage is not optional but rather mandatory. However, a procedure could include more than one tool and give the tester the option to select the most appropriate given the testing task to be performed.

A tool is a vehicle for performing a test process. The tool is a resource to the tester, but by itself is insufficient to conduct testing. For example, a hammer is a tool but until the technique for using that hammer is determined, the tool will lie dormant.

A testing technique is a process for ensuring that some aspect of an applications system or unit functions properly. There are few techniques but many tools. For example, a technique would be the advantage provided by swinging an instrument to apply force to accomplish an objective e.g., the swinging of a hammer to drive in a nail. The hammer is the tool used by swinging to drive in the nail. On the other hand, a swinging technique can also be used to split a log using an axe or to drive a stake in the ground with a sledgehammer.

The concept of tools and techniques is important in the testing process. It is a combination of the two that enables the test process to be performed. The tester should first understand the testing techniques and then understand the tools that can be used with each of the techniques.

There are many tools available to support the testing process, from simple checklists to defect tracking tools and automated regression tools. The Test Manager plays a key role in the identification, selection, and acquisition of automated tools. This section's discussion describes one acquisition process that may be used.

The management of any significant project requires the work be divided into tasks for which completion criteria can be defined. To permit orderly progress of the activities, the introduction of a test tool, and the scheduling of these events must be determined in advance. A general outline for such a schedule is discussed in "2.3.1.1 *Sequence of Events to Select Testing Tools*" on page 2-39. The actual calendar time schedule will depend on many factors, particularly on the time required for procurement of the tool and training. One format used for

the event sequence is consistent with the Critical Path Method (CPM) of project scheduling and can be used to develop the optimum calendar time schedule.

Most of the activities included in the event sequence are obviously necessary, but a few are included specifically to avoid the difficulties encountered when tools are obtained "through the side door" without adequate consideration of the resources required for the effective employment of the tool and without determination by a responsible manager that the tool will serve a primary need of the organization. Tools acquired without managerial approval are seldom used in an optimal way and are sometimes discarded. Experiences of this type are not conducive to gaining widespread acceptance of tools in the smaller programming environments where the activities required for the introduction of tools will impose, under the best of circumstances, a severe drain on resources. A key feature of the proposed approach is, therefore, that tool usage will be initiated only in response to an expressed management goal for software development or for the entire computing function.

Difficulties surrounding the introduction of tools can arise in three areas:

- Organizational obstacles
- Problems arising from the tools
- Obstacles in the computer environment

The individual activities described below, as well as the ordering of the event sequence, are designed to eliminate as many of these difficulties as possible. They are most effective with regard to organizational obstacles and probably least effective with regard to obstacles in the computer environment. The need for involving a responsible management level in the tool introduction has already been mentioned, and this is, indeed, the key provision for avoiding organizational obstacles. "Responsible management" is that level that has the authority to obligate the resources required for the introduction process.

The scope of the resource requirement will become clearer after all introduction activities have been described. Because one criterion for the selection of a tool is its ability to commit funds, the management level that can commit funds is hereafter referred to as funding management. In other organizations, this may be the project management, functional management, or department management.

The activities associated with the introduction of tools should include these activities:

- Identifying the goals to be met by the tool (or by the technique supported by the tool), and assigning responsibility for the activities required to meet these goals.
- Approving a detailed tool acquisition plan that defines the resource requirements for procurement and in-house activities.
- Approving the procurement of tools and training, if this is not explicit in the approval of the acquisition plan.
- Determining, after some period of tool use, whether the goals have been met.

The management of the organization that will introduce the tool must overcome additional organizational obstacles. A pitfall that must be avoided is assigning the details of the tool acquisition as a sideline to an individual who carries many other responsibilities. Even in a

small software organization (up to 14 programmers), it should be possible to make the tool introduction the principal assignment of an experienced individual with adequate professional background. This person is referred to as the Tool Manager. In medium-size organizations (15 to 39 testers), several individuals may be involved in deploying a test tool.

Obstacles arising from the tools themselves are expected to be avoided in the event sequence by a careful, methodical selection of tools. In particular, distinct contributions to the tool selection are specified for test management and the Test Manager.

Test management is assigned responsibility for:

- Identifying tool objectives
- Approving the acquisition plan (it may also require approval by funding management)
- Defining selection criteria
- Making the final selection of the tool or the source

The Test Manager is responsible for the following activities (note that a selection group for the more important test tools may assist the Tool Manager):

- Identifying candidate tools
- Applying the selection criteria (in informal procurement) or preparing the RFP (in formal procurement)
- Preparing a ranked list of tools or sources
- Conducting any detailed evaluations or conference room pilots

Further, the ultimate user of the tool is involved in the recommended event sequence in reviewing either the list of candidate tools or, for formal procurement, the tool requirements.

This distribution of responsibilities reduces the chances of selecting a tool that:

- Does not meet the recognized needs of the organization
- Is difficult to use
- Requires excessive computer resources
- Lacks adequate documentation

The repeated exchange of information required by the process outlined above will also avoid undue emphasis on very short-term objectives that may lead to the selection of a tool based on availability rather than suitability.

The obstacles to tool usage that reside in the computer environment are primarily due to the great diversity of computer architectures and operating system procedures, and to the lack of portability in most software tools. Activities associated with the introduction of tools can only modestly alleviate these difficulties.

2.3.1.1 Sequence of Events to Select Testing Tools

The event sequence provides the following help in this area:

1. Employ a methodical process of identifying candidate tools and selecting among these based on established criteria. This will avoid some of the worst pitfalls associated with "borrowing" a tool from an acquaintance or procuring one from the most accessible or persuasive tool vendor.
2. Determine the assignment and training of a Tool Manager who can make minor modifications to both the computer environment and the tool. This is expected to provide relief where there are version-related or release-related incompatibilities with the operating system, or where the memory requirements of the tool exceed the capabilities of the installation. In the latter case, remedies may be provided by removing tool options or by structuring the tool program into overlays.

The event sequence described below is conceived as a procedure generally applicable to the introduction of tools. For this reason, a systematic reporting of the experience with the introduction process as well as with the tool is desirable. The evaluation plan and the evaluation report specified in the event sequence support these goals.

2.3.1.1.1 Recommended Event Sequence

The event sequence described below is applicable to both small and large IT tool environments. Because of differences in tool requirements, personnel qualifications, and organizational structure, some differences in the content of the individual events will be expected. The event sequence addresses only the introduction of existing tools. Where a newly developed tool is introduced, a considerable modification of the activities and their sequence will be necessary.

The recommended event sequence allows for two procurement methods for bids:

1. Informal procurement (by purchase order)
2. Formal procurement (by a request)

Obviously, the latter is much more time-consuming, but it may lead to the procurement of better or cheaper tools. Acquisition of tools from the purchasing department or from government agencies should follow the informal procurement steps even when there is no procedural requirement for this. As mentioned above, tool acquisitions, which do not obtain the concurrence of all affected operational elements frequently, do not achieve their objectives.

Some steps are shown which can be combined or eliminated where less formal control is exercised or where plans or modifications required for the introduction of a tool are available from a prior user. The event sequence is intended to cover a wide range of applications. It was constructed with the thought that it is easier for the tool user to eliminate steps than to be confronted with the need for adding some that had not been covered in this section.

2.3.1.1.2 Event 1: Goals

The goals should be identified in a format that permits later determination that they have been met (see "2.3.1.1.15 *Event 14: Determine if Goals Are Met*" on page 2-47).

Typical goal statements are:

- Reduce the average test time by one-fifth.
- Achieve complete interchangeability of test data sets.
- Adhere to an established standard for documentation format.

The goals should also identify responsibilities, in particular, the role that project development may have and coordination with other activities. Where a decentralized management method is employed, the goals may include a not-to-exceed budget and a desired completion date. Once these constraints are specified, funding management may delegate the approval of the acquisition plan to a lower level.

2.3.1.1.3 Event 2: Tool Objectives

The goals generated in Event 1 are translated into desired tool features and requirements arising from the development and operating environment. Constraints on tool cost and availability may also be added at this event.

Typical test tool objectives are:

- The tool must run on our ABC computer under XOSnn.
- Only tools that have been in commercial use for at least one year and at no less than N different sites shall be considered.

At this point, the sequence continues with either Event 2: A or Event 2: B.

Event 2: A Acquisition Activities for Informal Procurement

Event 2: A1 – Acquisition Plan

The acquisition plan communicates the actions of test management both upward and downward. The plan may also be combined with the statement of the tool objectives created in Event 2. The acquisition plan should include:

- Budgets and schedules for subsequent steps in the tool introduction.
- Justification of resource requirements in light of expected benefits.
- Contributions to the introduction expected from other organizations (e.g., the tool itself, modification patches, or training materials).
- Assignment of responsibility for subsequent events within the IT organization, particularly the identification of the Test Manager.
- Minimum tool documentation requirements.

Event 2: A2 – Selection Criteria

The criteria should include ranked or weighted attributes that will support effective utilization of the tool by the user. Typical selection criteria are:

- Accomplishment of specified tool objectives
- Ease of use
- Ease of installation
- Minimum processing time
- Compatibility with other tools
- Low purchase or lease cost
- Documentation, Training, and Support availability

Most of these criteria need to be factored further to permit objective evaluation, but this step may be left up to the individual who does the scoring. Together with the criteria (most of which will normally be capable of a scalar evaluation), constraints, which have been imposed by the preceding events or are generated at this step, should be summarized.

Event 2: A3 – Identify Candidate Tools

This is the first event for which the Test Manager is responsible. The starting point for preparing a listing of candidate tools is a comprehensive tool catalog. A desirable but not mandatory practice is to prepare two lists:

- The first list contains all tools meeting the functional requirements without considering the constraints. For the viable candidates, literature should be requested from the supplier and examined for conformance with the given constraints.
- The second list contains tools that meet both the functional requirements and the constraints. If this list does not have an adequate number of entries, relaxation of some constraints will have to be considered.

Event 2: A4 – User Review of Candidates

The user(s) reviews the list of candidate tools prepared by the Test Manager. Because few users can be expected to be very knowledgeable in the software tools area, specific questions may need to be raised by the Tool Manager, such as: "Will this tool handle the present file format? Are tool commands consistent with those of the editor? How much training will be required?”

Adequate time should be budgeted for this review and a due date for responses should be indicated. Because the user views this as a far-term task, of lower priority than many immediate obligations, considerable follow-up by the Tool Manager will be required. If tools can be obtained for trial use, or if a

demonstration at another facility can be arranged, it will make this step much more significant.

Event 2: A5 – Score Candidates

For each of the criteria previously identified, a numerical score is generated on the basis of the following:

- Information obtained from a vendor's literature.
- Demonstration of the tool.
- The user's review.
- Observation in a working environment.
- Comments of prior users.

If weighting factors for the criteria are specified, the score for each criterion is multiplied by the appropriate factor and the sum of the products represents the overall tool score. Where only a ranking of the criteria was provided, the outcome of the scoring may be simply a ranking of each candidate under each criteria heading. Frequently, a single tool is recognized as clearly superior in this process.

Event 2: A6 – Select Tool

This decision is reserved for test management in order to provide review of the scoring, and to permit additional factors that were not expressed in the criteria to be taken into consideration. For example, a report might just have been received from another organization to which the selected vendor did not provide adequate service. If the selected tool did not score highest, the Tool Manager should have an opportunity to review the tool characteristics thoroughly to avoid unexpected installation difficulties.

The tool selection concludes the separate sequence for informal procurement for "*Event 2: B Acquisition Activities for Formal Procurement*." Continue with "Event 3: Procure Tool" on page 2-44.

Event 2: B Acquisition Activities for Formal Procurement

Event 2: B1 – Acquisition Plan

The plan generated here must include all elements mentioned under Event 2: A1, plus:

- The constraints on the procurement process.
- The detailed responsibilities for all procurement documents (statement of work, technical and administrative provisions in the request for proposal (RFP), etc.).

Event 2: B2 – Technical Requirements Document

The technical requirements document is an informal description of the tool requirements and the constraints under which the tool has to operate. It will utilize much of the material from the acquisition plan but should add enough detail to support a meaningful review by the tool user.

Event 2: B3 – User Review of Requirements

The user reviews the technical requirements for the proposed procurement. As in the case of Event 2: A4 – User Review of Candidates, the user may need to be prompted with pertinent questions, and there should be close management follow-up in order to get a timely response.

Event 2: B4 – RFP Generation

From the Technical Requirements Document and its user comments, the technical portions of the RFP can be generated. Usually these include:

- *Specification*

This should include applicable documents, a definition of the operating environment, and the quality assurance provisions.

- *Statement of Work*

This should state any applicable standards for the process by which the tool is generated (e.g., configuration management of the tool), and documentation or test reports to be furnished with the tool. Training and operational support requirements are also identified in the Statement of Work.

- *Proposal Evaluation Criteria and Format Requirements*

Evaluation criteria are listed in the approximate order of importance. Restrictions on proposal format (major headings, page count, and desired sample outputs) may also be included.

Event 2: B5 – Solicitation of Proposals

Administrative and purchasing personnel carry out this activity. Capability lists of potential sources are maintained by most purchasing organizations. Where the software organization knows of potential bidders, their names should be made known to the procurement office. When responses are received, they are screened for compliance with major legal provisions of the RFP.

Event 2: B6 – Technical Evaluation should be Consistent

Each of the proposals received in response to the RFP is evaluated against the criteria previously established. Failure to meet major technical requirements can lead to outright disqualification of a proposal. Those deemed to be in the “competitive range" are assigned point scores. These point scores are used together with cost and schedule factors that are being separately evaluated by administrative personnel.

Event 2: B7 – Source Selection

On the basis of the combined cost, schedule, and technical factors, a source for the tool is selected. If this was not the highest-rated technical proposal, prudent management will require additional reviews by test management and the Tool Manager to determine that it is indeed acceptable.

The source selection concludes the separate sequence for formal procurement for "2.3.1.1.3 *Event 2: Tool Objectives*." Continue with 2.3.1.1.4 *Event 3: Procure Tool* below.

2.3.1.1.4 Event 3: Procure Tool

In addition to determining whether the cost of the selected tool is within the approved budget, the procurement process also:

- Considers the adequacy of licensing and other contractual provisions and compliance with the "fine print" associated with all the organization's procurements.
- Identifies the vendor's responsibility for furnishing the source program, meeting specific test and performance requirements, and tool maintenance.

In informal procurement, a period of trial use may be considered if this has not already taken place under one of the previous events.

If the acquisition plan indicates the need for outside training, the ability of the vendor to supply the training and the cost advantages from combined procurement of tool and training should be investigated. If substantial savings can be realized through simultaneous purchase of tool and training, procurement may be held up until outside training requirements are defined. See "2.3.1.1.7 *Event 6: Training Plan*" on page 2-45).

2.3.1.1.5 Event 4: Evaluation Plan

The evaluation plan is based on the goals identified in "2.3.1.1.3 *Event 2: Tool Objectives*" and the tool objectives derived from these in "2.3.1.1.3 *Event 2: Tool Objectives*." It describes how the attainment of these objectives is to be evaluated for the specific tool selected. Typical items to be covered in the plan are:

- Milestones for installation
- Dates
- Performance levels for the initial operational capability and for subsequent enhancements
- Identify the reports and supporting data that address expected improvements in throughput, response time, or turnaround time
- Assign responsibility for tests, reports, and other actions
- A topical outline of the evaluation report

The procedure for the acceptance test is a part of the evaluation plan, although in major tool procurement it may be a separate document. It lists the detailed steps necessary to test the tool in accordance with the procurement, provisions when it is received, to evaluate the interaction

of the tool with the computer environment (e.g., adverse effects on throughput), and for generating an acceptance report.

2.3.1.1.6 Event 5: Implementation Plan

The plan will describe the responsibilities and tasks for the implementation of the tool, and the training that will be required. An experienced system programmer, familiar with the current operating system, should do the implementation. Training in the operation and installation of the selected tool in the form of review of documentation, visits to current users of the tool, or training by the vendor must be arranged.

2.3.1.1.7 Event 6: Training Plan

The training plan should first consider the training inherently provided with the tool, (e.g., documentation, test cases, online diagnostics, Help.) These features may be supplemented by standard training aids supplied by the vendor for Internet and in-house training such as audio or videocassettes and lecturers.

Because of the expense, training sessions at other locations should be considered only when none of the previous categories is available. The number of personnel to receive formal training should also be specified in the plan, and adequacy of in-house facilities (number of terminals, computer time, etc.) should be addressed. If training by the tool vendor is desired, this should be identified as early as possible to permit training to be procured with the tool (see “2.3.1.1.4 *Event 3: Procure Tool*” on page 2-44).

User involvement in the preparation of the training plan is highly desirable, and coordination with the user is considered essential. The training plan is normally prepared by the Tool Manager and approved by test management.

Portions of the plan should be furnished to procurement staff if outside personnel or facilities are to be utilized.

2.3.1.1.8 Event 7: Tool Received

The tool is turned over by the procuring organization to the Tool Manger or systems programmer.

2.3.1.1.9 Event 8: Acceptance Test

The Tool Manager and test staff test the tool. This is done as much as possible in an "as received" condition with only those modifications made that are essential for bringing it up on the host computer. A report on the test is issued. After approval by test management, it constitutes the official acceptance of the test tool.

2.3.1.1.10 Event 9: Orientation

When it’s determined that the tool has been received in a satisfactory condition, test management holds an orientation meeting for all personnel involved in the use of the tool and

tool products (reports or listings generated by the tool). The main purpose is to communicate as directly as possible the objectives of the tool use, such as increased throughput or improved legibility of listings.

Highlights of the evaluation plan should also be presented, and any changes in duties associated with the introduction of the tool should be described. Personnel should be reassured that allowance will be made for problems encountered during the introduction, and the full benefits of the tool may not make themselves felt for some time.

2.3.1.1.11 Event 10: Modifications

The systems programmer and Tool Manager carry out this step in accordance with the approved toolsmithing plan. It includes modifications of the following:

- The Tool

  Typical tool modifications involve deletion of unused options, changes in prompts or diagnostics, and other adaptations made for efficient use in the prevailing environment. Documentation of the modifications is an essential part of this event.

- Documentation

  Vendor literature for the tool is reviewed in detail and is tailored for the prevailing computer environment and for the tool modifications that have been made. Deleting sections that are not applicable can be just as useful as adding material that is required for the specific programming environment. Unused options shall be clearly marked or removed from the manuals. If there is some resident software for which the tool should not be used (e.g., because of language incompatibility or conflicts in the operating system interface), warning notices should be inserted into the tool manual.

- Operating system

  In rare cases some modification of the computer proper may also be necessary (channel assignments, etc.).

2.3.1.1.12 Event 11: Training

Training is a joint responsibility of the Tool Manager and the tool user. The former is responsible for the content (in accordance with the approved training plan), and the latter should have control over the length and scheduling of sessions.

Training is an excellent opportunity to motivate the user to utilize the tool. The tool user should have the privilege of terminating steps in the training that are not helpful and extending portions that are helpful and need greater depth. Training is not a one-time activity. Retraining or training in the use of additional options after the introductory period is desirable. This also provides an opportunity for users to talk about problems with the tool.

2.3.1.1.13 Event 12: Use in the Operating Environment

The first use of the tool in the operating environment should involve the most qualified test personnel and minimal use of options. The first use should not be on a project with tight

schedule constraints. Any difficulties resulting from this use must be resolved before expanded service is initiated. If the first use is successful, use by additional personnel and use of further options may commence.

User comments on training, first use of the tool, and use of extended capabilities are prepared and furnished to the Tool Manager. Desired improvements in the user interface, speed or format of response and utilization of computer resources are appropriate topics. Formal comments may be solicited shortly after the initial use, after six months, and again after one year.

2.3.1.1.14 Event 13: Evaluation Report

The Tool Manager prepares the evaluation report, using the outline generated in "2.3.1.1.5 *Event 4: Evaluation Plan*" on page 2-44. The report should include:

- User comments and observations of the systems programmer.
- Whether the general goals and tool objectives were met.
- Observations on the installation and use of the tool.
- Cooperation received from the vendor in installation or training.
- Any other "lessons learned."
- Tool and host computer modifications.
- A section of comments useful to future users of the tool.

The report is approved by test management and preferably also by funding management.

2.3.1.1.15 Event 14: Determine if Goals Are Met

Funding management receives the evaluation report and determines whether the goals established in "2.3.1.1.5 *Event 4: Evaluation Plan*" on page 2-44 have been met. This determination shall be in writing and include:

- Attainment of technical objectives.
- Adherence to budget and other resource constraints.
- Timeliness of the effort.
- Cooperation from other departments.
- Recommendations for future tool acquisitions.

2.3.2 Classes of Test Tools

"2.3.1.1.13 *Event 12: Use in the Operating Environment*" in the Tool Development and Acquisition Process involves using the tool in an operating environment. However, the preceding events prepare the organization for using the tool effectively, and the latter events assure that the appropriate tool has been selected.

There are literally hundreds of tools available for testers. Some are relatively simple and available at no charge on the Internet; others are very expensive and require extensive training and skills to be used effectively. Because tools change rapidly, older tools being deleted and new ones being added, testers cannot be expected to know the totality of test tools available in the marketplace.

While there are no generally accepted categories of test tools, experience has shown that the most commonly used tools can be grouped into these eight areas:

- Automated Regression Testing Tools

  Tools that can capture test conditions and results for testing new versions of the software.

- Defect Management Tools

  Tools that record defects uncovered by testers and then maintain information on those defects until they have been successfully addressed.

- Performance/Load Testing Tools

  Tools that can “stress” the software. The tools are looking for the ability of the software to process large volumes of data without either losing data, returning data to the users unprocessed, or have significant reductions in performance.

- Manual Tools

  One of the most effective of all test tools is a simple checklist indicating either items that testers should investigate, or to enable testers to ensure they have performed test activities correctly. There are many manual tools such as decision tables, test scripts to be used to enter test transactions and checklists for testers to use when performing such testing techniques as reviews and inspections.

- Traceability Tools

  One of the most frequently used traceability tools is to trace requirements from inception of the project through operations.

- Code Coverage

  Tools that can indicate the amount of code that has been executed during testing. Some of these tools can also identify non-entrant code.

- Test Case Management Tools

  This category includes test generators and tools that can manage data being processed for online assistance.

- Common tools that are applicable to testing

  Testers have access to a variety of work tools, many included with operating software such as “Windows.” These include such things as word processing, spreadsheets, computer graphics used for reporting and status checking, and tools that can measure the reading difficulty of documentation.

Most testing organizations agree that if the following three guidelines are adhered to tool usage will be more effective and efficient.

- Guideline 1

  Testers should not be permitted to use tools for which they have not received formal training.

- Guideline 2

  The use of test tools should be incorporated into test processes so that the use of tools is mandatory, not optional.

- Guideline 3

  Testers should have access to an individual in their organization, or the organization that developed the tool, to answer questions or provide guidance on using the tool.

2.4 Testers Competency

Test competency is a direct responsibility of the individual and the organization that employs that individual. However, the individual has the primary responsibility to ensure that his/her competencies are adequate and current. For example, if a tester today was testing Cobol programs, and that tester had no other skill sets than testing Cobol programs, the probability of long-term employment in testing is minimal. However, if that individual maintains current testing competencies by activities such as pursuing the CSTE designation, learning new testing tools and techniques, that individual is prepared for new assignments, new job opportunities and promotions.

Test competency is based on two criteria. The first is the skill sets possessed by that individual; for example, skills in writing test plans and using specific test tools. The second is performance; for example, how those skills are applied to real-world test situations. An individual may possess the skills necessary to write a test plan but when assigned to a testing project, can that individual actually write an effective and efficient test plan?

The certifications offered by the Software Certifications organization parallel these two competency criteria. The initial certification such as the CSTE is focused on skill sets. The advanced certifications such as the Certified Manger of Software Testing and the Certified Innovator of Software Testing are focused on the performance competency of the individual.

Software testing organizations should develop a road map for testers to pursue to advance their competencies. The road map will have two paths:

- Skill Sets

  This path will define the totality of skills that will be needed by the testing organization and the sequence in which those skills should be learned. For example, an individual might learn how to prepare test data before they learn how to write test plans.

- Performance Skills

  This path must show individuals that they must be able to perform on the job, and the sequence in which they must learn how to perform those tasks. For example, they must learn how to create effective test data before they can learn how to write effective and efficient plans.

The skill sets are evaluated primarily on academic standards. For example, “Have you gone to and received a certificate from a course on a specific testing tool?” Evaluating performance is usually based on results. For example, “Can you create X testing transactions within a specified time period?”

Figure 2-4 Measuring the Competency of Software Testers is typical of how a software testing organization may measure an individual tester’s competency. This type of chart is developed by Human Resource organizations to be used in performance appraisals. Based on the competency assessment in that performance appraisal, raises and promotions are determined.

Individuals should use their organization’s training paths to build their competencies for their specific organization. However, individuals should also use the CSTE CBOK as what might be necessary to obtain better positions and have greater software testing competencies.

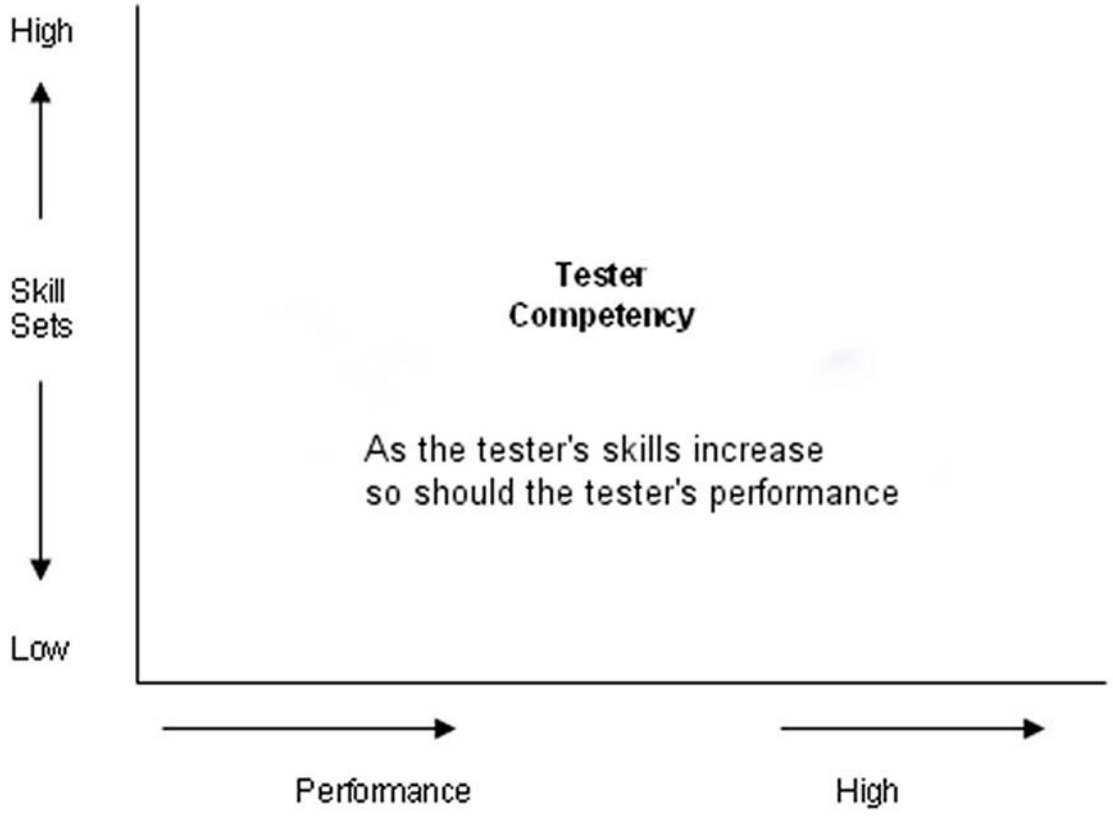


Figure 2-4 Measuring the Competency of Software Testers

Skill Category

3

Managing the Test Project

Software testing is a project with almost all the same attributes as a software development project. Software testing involves project planning, project staffing, scheduling and budgeting, communicating, assigning and monitoring work, and ensuring that changes to the project plan are incorporated into the test plan.

Test Administration	**3-1**
Test Supervision	**3-11**
Test Leadership	**3-29**
Managing Change	**3-36**

3.1 Test Administration

Test administration is managing the affairs of software testing. It assures what is needed to test effectively for a software project will be available for the testing assignment. Specifically this section addresses:

- Test Planning – assesses the software application risks, and then develops a plan to determine if the software minimizes those risks.
- Budgeting – the resources to accomplish the test objectives.
- Scheduling – dividing the test project into accountable pieces and establishing start and completion dates.
- Staffing – obtain the testers to achieve the plan.
- Customization of the test process – determining whether or not the standard test process is adequate for a specific test project, and if not, customizing the test process for the project.

Logically the test plan would be developed prior to the test schedule and budget. However, testers may be assigned a budget and then build a test plan and schedule that can be accomplished within the allocated budget. The discussion in this skill category will discuss planning, scheduling and budgeting as independent topics, although they are all related.

The five key tasks for test project administration are the planning, budgeting, scheduling, staffing, and customization of the test process if needed. The plan defines the steps of testing, the schedule determines the date testing is to be completed, the budget determines the amount of resources that can be used for testing and staffing, and the test process customization assures the test process will accomplish the test objectives.

Because testing is part of a system development project its plan, budget, and schedule cannot be developed independently of the overall software development plan, budget and schedule. The build component and the test component of software development need to be integrated. In addition, these plans, schedules and budgets may be dependent on available resources from other organizational units, such as user participation.

Each of these five items should be developed by a process: processes for developing test planning, budgeting, scheduling, staffing, and test process customization. The results from those processes should be updated throughout the execution of the software testing tasks. As conditions change so must the plan, budget, schedule, and test process change. These are interrelated variables. Changing one has a direct impact on the other three.

3.1.1 Test Planning

Test planning is a major component of software testing. It is covered in detail in Skill Category 4. That category provides the process for developing the test plan and provides a standard for defining the components of a test plan.

3.1.2 Budgeting

There is no one correct way to develop a budget. Some IT organizations use judgment and experience to build the budget; others use automated estimating tools to develop the budget.

The following discussion of budgeting represents some of the better budgeting processes, but not necessarily the only budgeting processes. The tester needs to be familiar with the general concept of budgeting and then use those processes available in their IT organization. Every project is unique. Different factors play a role in making one project differ from another. Among the factors that must be estimated are: size, requirements, expertise, and tools.

To a great degree, test management depends on estimation and judgment. By judgment we mean, the expertise of the test manager or person responsible for the estimation. Internal factors within the organization and external factors (such as economy and client requests) always affect the project. This is where risk analysis and estimate meet. Estimation involves risk at all levels. We can say, “The importance of estimation cannot be underestimated.”

Factors that influence estimation include, but are not limited to:

- Requirements
- Past data
- Organization culture
- Selection of suitable estimation technique
- Own experience
- Resources available
- Tools at our disposal

Remember, by definition an estimate means something that can change; and it will. It is a basis on which we can make decisions. For this reason the project manager must continually monitor the project estimate and revise the estimate as necessary. The importance of monitoring will vary depending on the project phase. Figure 3-1 Average Test Cost by SDLC Phases shows the estimated testing costs by phase, which probably will change during test execution.

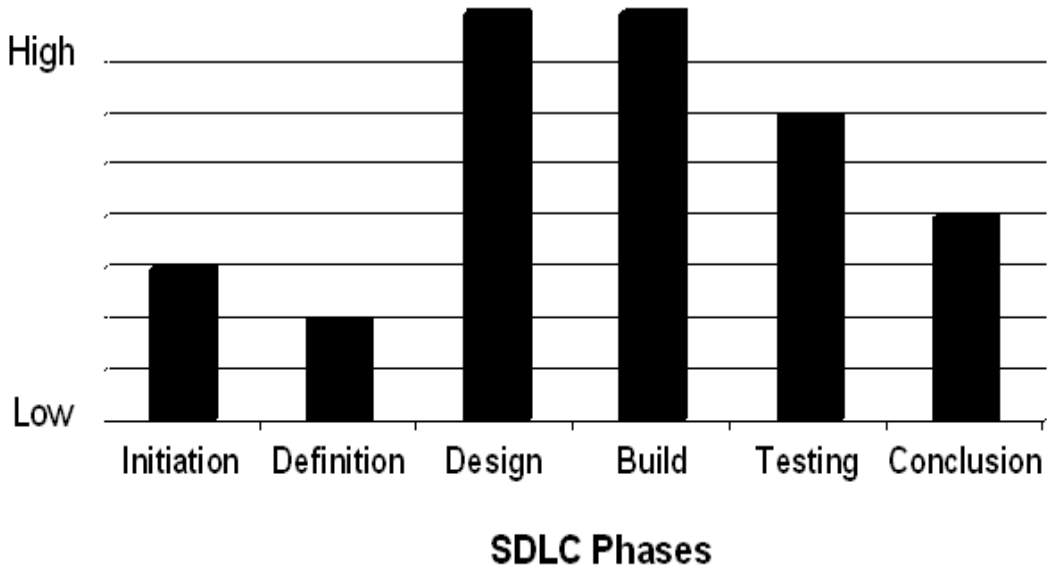


Figure 3-1 Average Test Cost by SDLC Phases

Estimating the budget and schedule for testing involves determining what effort and time will be needed to accomplish the testing goals as stated in the test plan. Accomplishing a testing goal is the objective of testing. In software testing, we are trying to achieve a goal. The Project Smart Web site (http://www.projectsmart.co.uk/smart_goals.html) has stated this very aptly. According to them, all goals should be smart goals wherein SMART stands for:

Specific

Well defined; clear to anyone that has basic knowledge of the project

Measurable

Know if the goal is obtainable and how far away completion is; know when it has been achieved

Agreed Upon

Agreement with all the stakeholders what the goals should be

Realistic

Within the availability of resources, knowledge and time

Time Frame

Enough time to achieve the goal; not too much time, which can affect project performance

When you estimate or run a project, take a moment to consider whether your testing goals are SMART goals. If they aren't, use the techniques mentioned above to re-write them.

3.1.2.1 Budgeting Techniques

Budgeting techniques are techniques to estimate the cost of a project. The following budgeting techniques are discussed below:

- Top-Down Estimation
- Expert Judgment
- Bottom-Up Estimation

3.1.2.1.1 Top-Down Estimation

The assumptions in the Top-Down Estimation approach are that software has its own complexity and difficulty in design and implementation. Project management uses this technique since they generate an overall estimate based on the initial knowledge. It is used at the initial stages of the project and is based on similar projects. Past data plays an important role in this form of estimation.

Preliminary estimates are required to determine the feasibility of a project and detailed estimates are needed to help with project planning. The choice of the model will depend on the purpose. There are several metrication methods and models in use. Size is a primary factor in costing models. We will briefly discuss the major ones. It is important to note that all models give the output based on the data input. Inaccurate or wrong inputs will give bad estimates.

The following types of models are used to estimate cost:

- Cost Models

  These models provide direct estimates of effort. They typically have a primary cost factor such as lines of code (LOC) and a number of secondary adjustment factors. Examples of cost models are Function Points and COCOMO.

- Constraint Models

These models demonstrate the relationship over time between two or more parameters of effort, duration, or resource. An example of a constraint model is the Putnam's SLIM model.

- Function Points Model

  Function points (FP) measure the size in terms of the amount of functionality in a system. Function points are computed first by calculating an unadjusted function point count (UFC) for different categories. They are then adjusted based on complexity to arrive at the Function Point. Function Points are useful when doing a feasibility analysis. It is always a good practice to have more than one person do the estimation. This way the organization can measure the subjectivity and also arrive at common ground in the future.

- COCOMOII Model

  COCOMOII is an enhancement over the original COCOMO (Constructive Cost Model). The original COCOMO model is based on inputs relating to the size of the system and a number of cost drivers that affect productivity. COCOMOII is useful for a wider collection of techniques and technologies. It provides support for object-oriented software, business software, software created via spiral or evolutionary development models and software using COTS application utilities.

3.1.2.1.2 Expert Judgment

If someone has experience in certain types of projects or certain tools, their expertise can be used to estimate the cost that will be incurred in implementing the project.

3.1.2.1.3 Bottom-Up Estimation

This cost estimate can be developed only when the project is defined as in a baseline. The WBS (Work Breakdown Structure) must be defined and scope must be fixed. The tasks can then be broken down to the lowest level and a cost attached to each. This can then be added up to the top baselines thereby giving the cost estimate. It gets its name 'bottom-up' since it works its way up from the bottom, which is the lowest level task.

At this level, each activity has a resource and skill level attached to it and dependencies have been identified. Contingencies have also been addressed.

3.1.2.2 Tracking Budgeting Changes

A cost estimate is an 'educated guess' of the costs that will be incurred. A budget is the accepted cost. A cost baseline is the comparison of the estimate versus actual cost. It is necessary to have a cost estimate for the following reasons:

- It helps in monitoring the cost with the goal of preventing the project from going beyond budget
- It helps to track variances allowing management to take appropriate action
- It helps in proving justification of costs incurred to the stakeholders

A cost baseline, once established, should not be changed unless approved as it is used to track variances against the planned cost during the execution of the project.

3.1.3 Scheduling

A schedule is a calendar-based breakdown of tasks and deliverables. It helps the project manager and project leader manage the project within the time frame and keep track of current, as well as future problems. A WBS (Work Breakdown Structure) helps to define the activities at a broader level, such as who will do the activities, but planning is not complete until we attach a resource and time to each activity. In simple terms, scheduling answers these questions:

- What tasks will be done?
- Who will do them?
- When will they do them?

A schedule requires constant update since teams and tasks undergo change. Status reports are the major input to the schedule. Scheduling revolves around monitoring the work progress versus work scheduled. A few advantages are:

- Once a schedule is made, it gives a clear idea to the team and the management of the roles and responsibility for each task
- It propagates tracking
- It allows the project manager the opportunity to take corrective action

A Work Breakdown Structure (WBS) groups test project components into deliverable and accountable pieces. As illustrated in Figure 3-2, it displays and defines the product to be developed by planning, training, and other elements and relates them to each other and to the product. The aim is to have measurable components for which people (team members, subject matter experts and specialists) can be held accountable. It is created during the proposal stage.

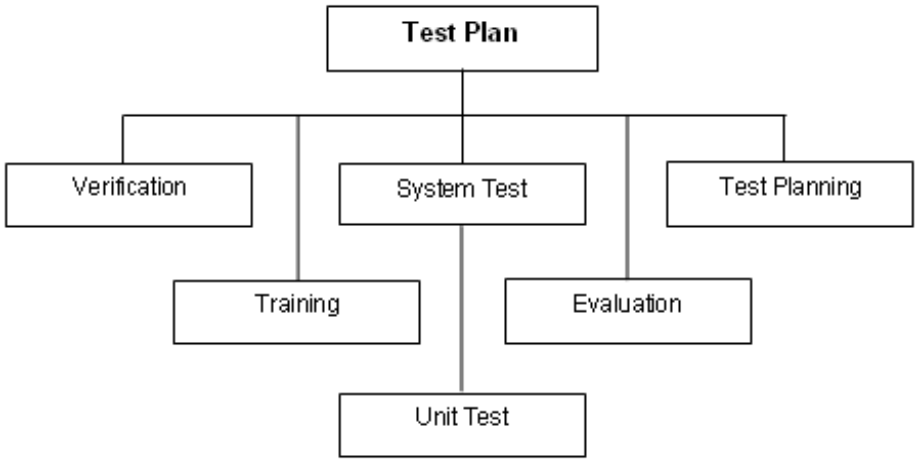


Figure 3-2 Example of a Work Breakdown Structure

The WBS defines the total project. It is a product-oriented, family tree composed of hardware elements, software elements, and service elements. The WBS relates project elements or work scope definitions to each other and to the end product. It is not an organization chart of company personnel.

There is a misconception that a WBS should be broken down to the smallest level. In actuality, it is only during subsequent detail planning that the WBS is broken down into measurable components which will then be further broken down into smaller and achievable components assigned to team members. As in budgeting, expert judgment is useful in scheduling

3.1.4 Staffing

Ideally, staffing would be done by identifying the needed skills and then acquiring members of the test project who possess those skills. It is not necessary for every member of the test team to possess all the skills, but in total the team should have all the needed skills. In some IT organizations, management assigns the testers and no determination is made as to whether the team possesses all the needed skills. In that case, it is important for the test manager to document the needed skills and the skills available by the team members. Gaps in needed skills may be supplemented by such individuals assigned to the test project on a short-term basis.

The recommended test project staffing matrix is illustrated in Table 3-1. This matrix shows that the test project has identified the needed skills. In this case they need the planning, test data generation skills, and skills in using tools X and Y. The matrix shows there are four potential candidates for assignment to that project. Assume that only two are needed for testing, the test manager would then attempt to get the two that in total had all the four needed skills.

If the test team does not possess the necessary skills, it is the responsibility of the test manager to teach those individuals the needed skills. This training can be on-the-job training, formal classroom training, or e-learning training.

Table 3-1: Test Project Staffing Matrix

	Skills Needed			
Staff	**Planning**	**Test Data Generation**	**Tool X**	**Tool Y**
A	**X**		**X**	
B		**X**	**X**	**X**
C	**X**			**X**
D		**X**	**X**	**X**

3.1.4.1 Test Team Approaches

The following four different approaches are used to build a test team:

1. Developers become the Test Team Approach
2. Independent IT Test Team Approach
3. Non-IT Test Team Approach
4. Combination Test Team Approach

3.1.4.1.1 Developers Become the Test Team Approach

The members of the project development team become the members of the test team. In most instances, the systems development project leader is the test team project leader. However, it is not necessary to have all of the development team members participate on the test team, although there is no reason why they would not participate. It is important that one member of the test team be primarily responsible for testing other member's work. The objective of the team is to establish a test process that is independent of the people who developed the particular part of the project being tested.

The advantage of the developers test team approach is that it minimizes the cost of the test team. The project is already responsible for testing, so using project members on the test team is merely an alternate method for conducting the tests. Testing using the developers test team approach not only trains the project people in good test methods, but also cross-trains them in other parts of the project. The developers test team approach uses those people in testing who are most knowledgeable about the project.

The disadvantage of the developers test team approach is the need for ensuring that the project team allocates appropriate time for testing. In addition, the project team members may lack team members who believe that the project solution is incorrect and thus find it difficult to challenge the project assumptions.

3.1.4.1.2 Independent IT Test Team Approach

Testing performed by IT personnel independently of the project does not relieve the project personnel of responsibility for the correctness of the application system. The independent testing is designed to provide a different perspective to testing in order to provide extra assurance of the correctness of processing. The independent testing normally occurs after the project team has performed the testing they deem necessary (i.e., unit testing). Frequently, the system development team verifies that the system structure is correct and the independent test team verifies that the system satisfies user requirements.

Independent testing is normally performed by either information services quality assurance or a professional testing group in the IT department. While the project team is involved in all aspects of the development, the quality assurance professional test teams specialize in the testing process. However, most individuals in these testing groups have had systems design and programming experience.

The advantage of independent information services is the independent perspective they bring to the test process. The group is comprised of information services professionals who have specialized in the area of testing. In addition, these groups have testing experience in multiple projects, and thus are better able to construct and execute tests than those individuals who only test periodically.

The disadvantage of independent IT testing is the additional cost that may be required to establish and administer a testing function. Also, the development team may place too much reliance on the test team and thus fail to perform adequate testing themselves, resulting in overburdening the professional testers. In addition, the competition between the test team and the project team may result in a breakdown of cooperation, making it difficult for the test team to function properly.

3.1.4.1.3 Non-IT Test Team Approach

Groups external to the information services department can perform testing. The three most common groups that test application systems are users, auditors, and consultants. These groups represent the organizational needs and test on behalf of the organization. They are concerned with protecting the interest of the entire organization.

The advantage of a non-IT test team is that they provide an independent view and at the same time can offer independence in assessment. Loyalty, or charter, to report unfavorable results to only the information services department, does not restrict the non-IT group. The non-IT group has greater ability to act and to cause action to occur once problems are detected than does a group within an information services department.

The disadvantage of non-IT testing is the cost of the test. Generally, these groups are not familiar with the application and must first learn the application and then learn how to test within the organization. The non-IT group may encounter difficulties in testing due to lack of knowledge of IT’s test environment and the project.

3.1.4.1.4 Combination Test Team Approach

Any or all of the above groups can participate on a test team. The combination team can be drawn together to meet specific testing needs. For example, if the project had significant financial implications, an auditor could be added to the test team; if it had communication concerns a communication consultant could be added.

The advantage of drawing on multiple skills for the test team is to enable a multi-disciplined approach to testing. In other words, the skills and backgrounds of individuals from different disciplines can be drawn into the test process. For some of the test participants, particularly users, it can be an educational experience to make them aware of both the system and the potential pitfalls in an automated system. In addition, a combination test team has greater clout in approving, disapproving, or modifying the application system based upon the test.

The disadvantage of the combination test team is the cost associated with assembling and administering the test team. It also may pose some scheduling problems determining when the tests will occur. Finally, the diverse backgrounds of the test team may make the determination of a mutually acceptable test approach difficult.

3.1.5 Customization of the Test Process

Skill Category 2 discussed assessing the overall test process with software development project goals and implementation process. This assessment is to assure that the test process will accomplish the test objectives, or whether the test process will need some customization to accomplish the test objectives. Some of the characteristics of software development that may cause customization of the test process are:

- Release cycle schedules
- Software development methodology
- User schedules
- Project status reporting
- Interfacing with other projects
- Interfacing with enterprise-wide databases
- Assuring the same naming conventions/data definitions are used for testing as for other projects

Test process customization can occur many ways, but ideally the customization process is incorporated into the test processes, primarily the test planning process. If not incorporated into the test process, customization should occur as a task for the test manager to perform.

The customization process may include any of the following:

- Adding new test tasks
- Deleting some test tasks currently in the test process
- Adding or deleting test tools

- Supplementing skills of assigned testers to assure the tasks in the test process can be executed correctly

3.2 Test Supervision

Supervision relates to the direction of involved parties, and oversight of work tasks to assure that the test plan is completed in an effective and efficient manner. Supervision is a combination of the supervisor possessing the skill sets needed to supervise, and the tasks that contribute to successful supervision.

There are literally thousands of books written on how to supervise work. There is no one best way on how to supervise a subordinate. However, most of the recommended approaches to supervision include the following:

- Communication skills

  The ability of the supervisor to effectively communicate the needed direction information, and resolution of potential impediments to completing the testing tasks effectively and efficiently.

- Negotiation and complaint resolution skills

  Some specific skills needed to make a supervisor effective, like resolving complaints, using good judgment, and knowing how to provide constructive criticism.

- Project relationships

  Developing an effective working relationship with the test project stakeholders.

- Motivation, Mentoring, and Recognition

  Encouraging individuals to do their best, supporting individuals in the performance of their work tasks, and rewarding individuals for effectively completing those tasks.

3.2.1 Communication Skills

Most studies show that the single most important skill possessed by a supervisor is the ability to communicate. Those supervisors who are effective communicators are almost always effective supervisors.

There are four categories of communication skills that are important for supervisors:

- Written and oral communication
- Listening Skills
- Interviewing Skills
- Analyzing Skills

3.2.1.1 Written and Oral Communication

An important aspect of supervision is the ability to communicate with other parties. An effective way to quickly improve the ability to communicate is to view every communication opportunity as making a proposal to another person. The proposal can be made orally, or it can be in a written document. The following discussion on making an oral proposal is much more elaborate than would be required for a supervisor communicating assignments or information to a subordinate, but the concepts are applicable to all oral and written communications.

Making a presentation is a marketing opportunity. Your customer is ready and eager to hear your solution to their problem. Proper preparation and presentation will create the proper environment for the acceptance and the successful implementation of your proposed solution.

Some general guidelines to follow as you make a presentation are:

- Emphasize that you are presenting the best solution to the customer's problems.
- Emphasize that your project team is well equipped to implement this solution.
- Sell the corporate experience of your project staff and yourself.
- Sell your personal management capabilities.
- Sell the technical expertise of the project staff and yourself.
- Sell your enthusiasm to do this project effectively, efficiently, and economically.

There are three main parts to the presentation:

- Preparing for the proposal
- Presenting the proposal
- Closing the proposal.

3.2.1.1.1 Preparing the Proposal

Preparation is very important for any proposal or oral presentation. Follow these recommended steps when preparing your proposal.

1. Outline your proposal:
   - The problem to be solved – A concise description of the customer's problem that will be solved by the proposal.
   - System proposal solution constraints - Any constraints that might have an impact on the solution, such as time, people, or budget.
   - The proposed solution - The solution to the customer's problem.
   - Impact on people - How the system will affect people; the type of people needed to use the system.
   - Impact on cost - The cost versus benefits.
   - How to approve the project (i.e., close) - What is necessary for approval to proceed from this point?

2. Prepare visual aids for the presentation. The visual aids lend themselves to a more informal discussion, provide the opportunity to switch back and forth between the visual aids, and can be used to answer questions. Some guidelines on preparing visual aids are:
   - Lay out your presentation, identifying key points to be covered first, and then develop visual aids to support those points. Do not make the all-too-common mistake of sketching out or selecting illustrations first, then trying to decide what point they make.
   - Use one visual aid for each major point in your presentation.
   - Use pictorial illustrations wherever possible.
   - Limit the text on each to no more than 25 words.
   - Each visual aid must make one, and only one, point.
   - Leave the details to your oral discussion and not the visual aid.
   - Your presentation should expand on areas included on the visual aids.
3. Emphasize in your mind the three most important points you want to make in the presentation. Be sure they are well covered. Use the old philosophy of "tell them what you are going to tell them, tell them, and then tell them what you have told them.” This should focus on the three most important points of your presentation.
4. Rehearse your presentation in front of your colleagues. If the proposal is worth making, it is worth rehearsing. You might want to rehearse several times to ensure that you can do it effectively. Urge your colleagues to ask questions to ensure that you can back up your facts.
5. Be sure you understand the alternatives that were considered for the solution. You should present one proposal only, but be prepared to address other solutions and why you did not accept them, as part of the questioning at the proposal meeting.
6. The presentation should not last more than one hour. If you are unable to cover the material in one hour, then take a break and continue again. The ability to concentrate for more than one hour is extremely limited.
7. It is better to say too little than too much. Generally, the presentation flows better if you say too little and have the audience ask questions, than to say too much making them bored. More sales are lost from saying too much than from saying too little.

3.2.1.1.2 Presenting the Proposal

If you are accustomed to speaking before a group, you know the ropes. On the other hand, if you have not had the opportunity to speak before groups of people in the past, the following hints might be useful:

- Start and finish your discussion on time.
- Make the contents of your discussion serve your customers' needs, not yours. Every word you say should be for their benefit, not yours.
- When preparing your briefing and when actually giving the presentation, think in broad concepts as opposed to specific details. If you have properly prepared, the details will come automatically.

- Be enthusiastic and act enthusiastically. Move around the front of the room. Maintain eye contact with your audience. Remember, enthusiasm is infectious.
- Use terms that your audience members will understand.
- Use terms that you understand. Do not use technical jargon just because your technical gurus used it in the proposal. Don't be afraid to admit that your technical people "handle those matters." Just make sure that one of them is in the room to answer the questions.
- Include examples to support all points covered. Remember, examples, not proof. Customers like to hear, "This specific problem can be solved using such-and-such a technique, as we discovered when implementing a similar system for so and so."
- Issue handouts summarizing your briefing, but only after you are done talking. Keep their attention on you, not on handouts, during your presentation.
- If you feel a bit nervous, have someone else prepare a short (10-15 minutes) discussion of some narrow aspect of the proposal (maintenance, a technical detail, installation of equipment, etc.). After you have talked a while, introduce the topic with, "Mr. Smith wrote this portion of the proposal and can explain the concepts much more eloquently than I." This will give you a breather to clear your throat, gather your wits, and dry your sweaty palms.
- Always have a cup of coffee or a glass of water by your side. If someone asks you a tough question take a sip of coffee or water while you collect your thoughts, then answer.
- It's not a sin to not know the answer to a question. Simply tell the questioner that you will get back to him or her later that day, or have someone else in the room answer. Many proposal evaluation teams include technical people who are not happy until they prove they know more than you do. Those people get their satisfaction either by catching you in an outright error, or by getting you to admit that they thought of a question that you could not answer. Guess which scenario they prefer.
- Finally, always end any briefing with impact. The last sentence you say is probably the most important.

3.2.1.1.3 Closing the Proposal

An important part of making a proposal is the closing. The close is getting approval to proceed according to the proposal. You should have prepared and included as part of the proposal the documentation necessary to proceed.

The closing will normally occur at the end of the presentation. For example, at the conclusion of the presentation, you might state to the highest-ranking customer in attendance, "If you will sign this approval document, we can begin on the project tomorrow."

Be prepared for the close at any time during the presentation. For example, if you get a clue from the customer the project is wanted, you might ask, "Do you have enough information to make a decision on the project?" Again, many sales have been lost because of too much information, rather than too little.

Don't be concerned about objections to your proposal; these should be considered. Objections are usually opportunities to close. For example, if the customer objects to it taking 18 months, you might be able to counteroffer with having the first part of the project up and running in 12 months. You should have considered these objections in your preparation, and be prepared to counteroffer during the presentation.

The purpose of the proposal from the producer perspective is to get agreement for more work. Nothing else should be considered a desired conclusion to the proposal. Push hard for a closing or you may never get one.

3.2.1.2 Listening Skills

Throughout school, students are taught the importance of speaking, reading, writing, and arithmetic, but rarely is much emphasis placed on listening. The shift in society from industrial production to information management emphasizes the need for good listening skills. This is particularly true in the practice of software testing – oral communication (which includes listening) is rated as the number-one skill for the tester.

Some facts about listening include:

- Many Fortune 500 companies complain about their workers' listening skills.
- Listening is the first language skill that we develop as children; however, it is rarely taught as a skill. Thus, in learning to listen, we may pick up bad habits.
- Listening is the most frequently used form of communication.
- Listening is the major vehicle for learning in the classroom.
- Salespeople often lose sales because they believe talking is more important than listening (thus, in ads a computer company emphasizes that they listen).

It is also important to understand why people do not listen. People do not listen for one or more of the following reasons:

- They are impatient and have other stimuli to respond to, such as random thoughts going through their mind.
- They are too busy rehearsing what they will say next, in response to someone.
- They are self-conscious about their communication ability.
- External stimuli, for example, an airplane flying overhead, diverts their attention.
- They lack the motivation and responsibility required of a good listener.
- The speaker’s topic is not of interest to them.

The listener must be aware of these detriments to good listening, so they can recognize them and devote extra attention to listening.

3.2.1.3 The 3-Step Listening Process

The listening process involves three separate steps:

1. Hearing the speaker
2. Attending to the speaker
3. Understanding the speaker.

The practice of listening requires these three listening steps to occur concurrently. Mastering each of these steps will help improve your listening abilities.

3.2.1.3.1 Step 1: Hearing the Speaker

Hearing the speaker requires an understanding of the five channels of communication incorporated into speech. Much of listening occurs beyond merely hearing the words. Let's look at the five channels through which a speaker delivers information to his/her audience:

- Information Channel - The speaker's subject.
- Verbal Channel - The words used by the speaker.
- Vocal Channel - The tone of voice associated with the various words.
- Body Channel - The body movements and gestures associated with the information being conveyed.
- Graphic Channel - The pictures, charts, etc. that the speaker uses to emphasize or illustrate the material being discussed.

Speakers normally use the information, verbal, vocal, and body channels in speaking. In some instances, they also use the graphic channel. Listening requires that there is a meeting of the minds on the information channel. Speakers sometimes skip around to different subjects, making it easy to lose the subject being covered on the information channel. In "Step 2: Attending to the Speaker," we discuss the importance of feedback to confirm the subject being covered on the information channel.

The vocal and body channels impact the importance of the verbal channel. The verbal channel includes the choice of words used to present information, but the vocal and body channels modify or emphasize the importance of those words. For example, the words in the verbal channel may be, "John says he can do it." However, the tone of the vocal channel might indicate that John cannot do it, or the use of a thumbs-down body channel signal will also indicate that John cannot do it.

Hearing the speaker involves an awareness of all five channels, and listening to and watching the speaker to be sure we are receiving what the speaker is saying through all five channels. To master the hearing step, you must pay attention to all five channels. If you miss one or more of the channels, you will not hear what the person is saying. For example, if you are only paying partial attention to the speaker when the words, "John can do it" are stated, you may hear that John can do it, while the speaker said that John could not do it.

3.2.1.3.2 Step 2: Attending to the Speaker

Attending to the speaker is sometimes referred to as being an active listener. Devote your full attention to the speaker to confirm that what you heard is what the speaker intended you to hear. You must first understand yourself and your situation. You must evaluate your motivation for wanting to listen to this speaker. If the subject is important to you, but the speaker is boring, it will require significantly more effort on your part to be a good listener.

The most important part of attending to the speaker is establishing an active listening ability. Active listening involves a lot of response and dynamics. Some people view the listening process as a passive skill where you sit back and let the other person talk. This is fine for hearing the speaker, but not for confirming what the speaker has said. Feedback is very important to the listening process, particularly in this step. Feedback can be a nonverbal response, such as nodding your head, or a verbal response such as a question or a statement of confirmation.

It is very important to send the right type of feedback to the speaker. The wrong type of feedback not only doesn’t confirm what the speaker said, but also can reduce or terminate the listening process. It is very irritating to a speaker who is providing information to have the listener stray from the subject. For example, the speaker might be describing a quality problem, and the listener changes the subject and asks where the speaker is going to have lunch that day.

Some suggestions to help in attending to the speaker are:

- Free your mind of all other thoughts and concentrate exclusively on the speaker's communication.
- Maintain eye contact with the speaker for approximately 80 percent of the time.
- Provide continuous feedback to the speaker.
- Periodically restate what you heard the speaker say, and ask the speaker to confirm the intent of the information spoken.
- Move periodically to the understanding step to ensure that the information passed has been adequately understood.

3.2.1.3.3 Step 3 - Understanding the Speaker

There are five types of listening. While people can listen several different ways concurrently, normally listening is limited to one of the five types. The type chosen will have an impact on the ability to understand what the speaker is saying. When one has deciphered the information channel (i.e., what the subject is) and related the importance of that subject to the audience, listening must be adjusted to ensure that we get the message we need.

The five types of listening and their impact on understanding are:

- Type 1: Discriminative Listening

  Directed at selecting specific pieces of information and not the entire communication. For example, one may be listening to determine if an individual did a specific step in

the performance of a task. To get this, listen more to the nonverbal expressions rather than the verbal channel.

- Type 2: Comprehensive Listening

  Designed to get a complete message with minimal distortion. This type of listening requires a lot of feedback and summarization to fully understand what the speaker is communicating. This type of listening is normally done in fact gathering.

- Type 3: Therapeutic Listening

  The listener is sympathetic to the speaker's point of view. During this type of listening, the listener will show a lot of empathy for the speaker's situation. It is very helpful to use this type of listening when you want to gain the speaker's confidence and understand the reasons why a particular act was performed or event occurred, as opposed to comprehensive listening where you want to find out what has happened.

- Type 4: Critical Listening

  The listener is performing an analysis of what the speaker said. This is most important when it is felt that the speaker is not in complete control of the situation, or does not know the complete facts of a situation. Thus, the audience uses this type of understanding to piece together what the speaker is saying with what has been learned from other speakers or other investigation.

- Type 5: Appreciative or Enjoyment Listening

  One automatically switches to this type of listening when it is perceived as a funny situation or an explanatory example will be given of a situation. This listening type helps understand real-world situations.

One must establish which type of understanding is wanted and then listen from that perspective.

3.2.1.4 Interviewing Skills

A software tester will use interviewing skills for many different purposes. The obvious one is interviewing an individual for the job of a software tester, or to be assigned to a specific software project. However, interviewing skills are also used for gathering data for test purposes. The tester may interview a user/customer to better understand how their job is performed, the tester may need to interview project development personnel to understand the structure and function of the software systems, and a tester may need to interview a subject matter expert such as an auditor to better understand the attributes of an effective system of internal control.

The primary purpose of interviewing is fact-finding. A second purpose is to convey information to the individual being interviewed. Interviewing involves oral communication, it involves listening skills, and it involves fact-finding. Oral communication and listening skills have previously been discussed in this section.

Fact-finding is a process of identifying facts which is a statement of a condition. In other words, a fact is some attribute of a condition that is agreed by involved parties to be correct. A fact could be the result of a test.

A finding is identifying a difference between what is and what should be. To obtain a finding you must know what the condition of an event should be. It is for this reason we talk about testable requirements which pre-define what the processing result should be.

If a processing result should be that individuals are paid time and a half over 40 hours of work, and a test result showed that individuals were not paid time and a half over 40 hours, that would be a fact. The finding would be the difference, that is, that people should have been paid time and a half but they were not paid time and a half.

When documenting a finding it should include:

- A fact – tells why the difference between what is and what should be is significant.
- Cause – tells the reasons for the deviation. Identification is necessary as a basis for corrective action.
- Significance – how important the difference is in the context of the testing assignment.

3.2.1.5 Analyzing Skills

The first question asked after receiving a finding is: “What should I do about it?” The answer to that question is a recommendation. A recommendation suggests the action that should be taken to resolve a finding.

Findings and recommendations are two important parts of communication. The finding states what has happened, and the recommendation states what to do about it. Time spent carefully constructing recommendations are normally rewarded by increased acceptance of the recommendation.

Developing recommendations requires analysis. Unfortunately the effects of poor analysis are not as apparent as those of poor grammar or spelling. Poor analysis, however, is more destructive to the review process. Analysis relies on facts and inferences. The recipient of the report has questions about how and what occurred. These are best answered by the facts, but the question why, the most important question, must be answered by inference or by conclusions based on facts.

The individual developing a recommendation is aware that his/her conclusion is a judgment based on a preponderance of evidence and seldom is an absolute, inevitable determination. Much of the frustration occurring in getting a recommendation accepted can be traced directly to this awareness.

Analysis is an essential part of the software tester’s job. A simple recitation of facts, no matter how solid the facts are, creates questions in the mind of the recipient. When recommendations are made, the recipient asks other questions, such as, “How adequate are the criteria backing the recommendation?” “Will the recommendation cause more problems, or cost more, than the current method?” “How sound is the analysis?” “How effective is the recommendation?”

Sharing and exploring both facts and analysis helps to establish the value of the recommendation for the recipient of the recommendation.

Recommendations are based upon findings using the analysis process. Analysis permits the findings and the supporting background information to be subjected to a challenging process in order to develop recommendations. The value of the recommendation is normally related to the thoroughness of the analysis process.

The analysis process is illustrated in Figure 3-3. The figure shows that the problems plus the analysis produce recommendations. The problems or findings are normally a combination of facts, opinions, and circumstances. Analysis, on the other hand, is a scientific process used to produce a result or recommendation.

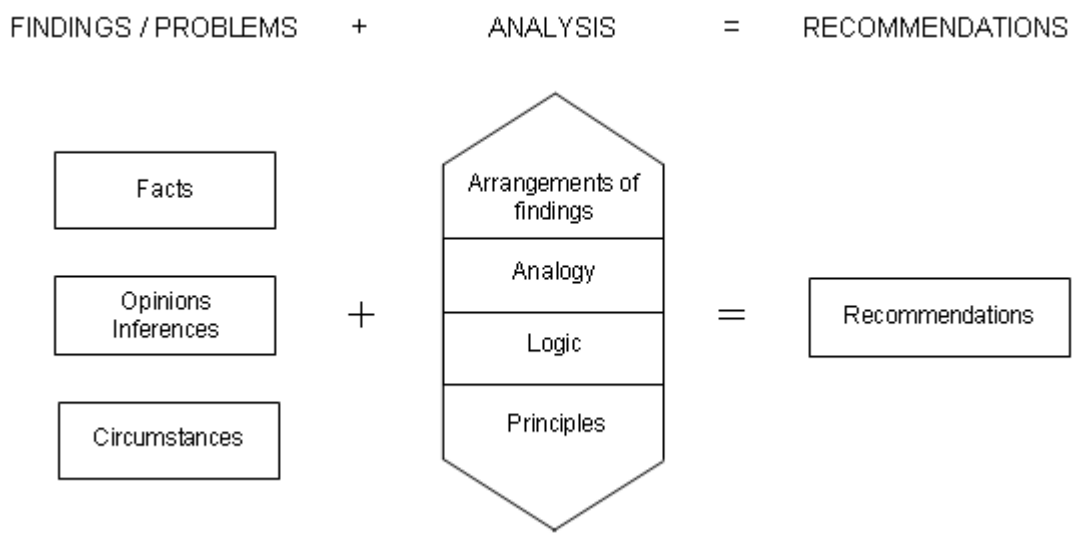


Figure 3-3 The Analysis Process

There are four general methods used for analysis which are:

- Arrangements

  The facts, opinions, and circumstances are arranged to enable relations and patterns to be shown between the facts. The relationship can be used to demonstrate cause and effect as well as correlations. For example, if the facts were arranged to show that there was a direct correlation between extent of training and number of errors, a recommendation could be built on that correlation.

  A simple method to arrange and rearrange facts is to code them using a simple coding method. Normally, any one fact will have several different codes. For example, an error condition might be coded as follows:

  - Input data entry error
  - Computer system error

- Accounts receivable error

The facts and opinions can then be arranged and rearranged in a variety of sequences to show patterns and relationships between the information.

- Analogy

  Using the analogy method, one situation is compared with or contrasted to another. This makes heavy use of the reviewer's judgment and experience. The reviewer, drawing upon his/her experience, utilizes the similarity between situations in an effort to capitalize on previous situations and recommendations which are applicable to the current situation.

- Logic

  The reviewer can use inductive or deductive logic to develop a recommendation. Using inductive logic, the argument moves from facts to a generalization. The generalization then becomes the situation that needs to be addressed by the commendation. Using deductive logic, the main idea is stated and then supported by the facts. Using this approach, the commendation is obvious and only needs to be justified by the facts in the situation.

- Principles

  The reviewer can rely upon good business practices and principles. These principles dictate the best method to accomplish tasks. When it can be determined from the analysis process that good businesses practice or principle has been violated, and thus caused a problem, the recommendation is the reinstatement of the principle. For example, if the problem is diagnosed as high maintenance costs and the analysis process shows that the principle of "formal systems documentation" has not been followed, the recommendation would be to document the system using a formal documentation.

3.2.2 Negotiation and Complaint Resolution Skills

As is the case with many activities, negotiating a complaint to resolution is a skill that is learned through practice. In many situations it is prudent to document the steps you take toward resolution and you must be willing to take actions that are appropriate to every level of the negotiation.

3.2.2.1 Negotiation

Conflict can be defined as a breakdown in the decision-making process. An acceptable decision cannot be made among the alternate positions available. Understanding the root cause of the conflict is the first step in resolving the conflict. Negotiation is the means by which the conflict will be resolved. The sources of conflict are listed and defined in Table 3-2.

Table 3-2: Conflict in Project Environments

Sources of Conflict	Definitions
Project Priorities	Views of project participants differ over sequence of activities and tasks.
Administrative Procedures	Managerial and administration-oriented conflicts over how the project will be managed.
Technical Opinions and Performance Trade-Offs	Disagreements over technical issues, performance specifications, technical trade-offs.
Human Resource	Conflicts about staffing a project team with personnel from other areas.
Cost	Conflict over cost estimates from support areas regarding work breakdown structures.
Schedule	Disagreements about the timing, sequencing, and scheduling of project-related tasks.
Personality	Disagreements on interpersonal issues.

In determining the root cause of the conflict, a supervisor needs to use all of the communication skills. These will help define the root cause of the conflict. Once the root cause has been identified, a decision-making process to eliminate the problem can commence. However when a conflict is interpersonal, it may be more difficult to find the root cause, and thus more difficult to resolve the conflict. Conflict resolution is a subset of conflict management. Conflicts are usually solved in one of these ways:

- Forcing

  Conflict is resolved when one party is successful in achieving its own interests at the expense of the other party's interest through the use of high relative power. This is the win-lose approach.

- Withdrawal

  Conflict is resolved when one party attempts to satisfy the concerns of others by neglecting its own interests or goals.

- Smoothing

  An unassertive approach – Both parties neglect the concerns involved by sidestepping the issue or postponing the conflict or choosing not to deal with it.

- Compromise

  An intermediate approach – Partial satisfaction is sought for both parties through a "middle ground" position that reflects mutual sacrifice. Compromise evokes thoughts of giving up something, therefore earning the name "lose-lose."

- Problem-solving

  Cooperative mode – Attempts to satisfy the interests of both parties. In terms of process, this is generally accomplished through identification of "interests" and freeing the process from initial positions. Once interests are identified, the process

moves into a phase of generating creative alternatives designed to satisfy interests (criteria) identified.

The conflict resolution methods of withdrawal and smoothing may temporarily address conflict but fails to resolve the root cause. If the conflict resolution approach is withdrawing, one party loses and one party wins. If the conflict resolution is smoothing, both parties lose.

The conflict resolution methods of forcing, compromising and problem-solving resolve the conflict and reach a decision. However, in forcing the decision, one party wins and another party loses. Compromising only to provide resolution may mean that both parties lose. The win-win conflict resolution process is problem-solving. Negotiations in this conflict resolution approach will assure that the most important interests of both parties are achieved so that each party feels they win.

3.2.2.2 Resolving Complaints

Research shows that complaints must be resolved within four minutes. Within that time, you should be communicating the solution of the problem to the customer. In his book, Contact: The First Four Minutes, Dr. Leonard Zunin, a human relations consultant, states that unless you have satisfied your customer within four minutes, they will give up on you. They will sense that you have not accepted the urgency of their problem and that you are not the one to solve their problem.

If you make your customer's problem your problem – in other words, you sincerely want to resolve his or her complaint – then you need to execute the following process.

3.2.2.3 The 4-Step Complaint-Resolution Process

3.2.2.3.1 Step 1: Get On Your Customer’s Wavelength

You cannot begin to resolve your customer's complaint until you show your customer your concern for their problem. You need to:

- Get on the same physical wavelength. Establish a position for mutual discussion. If your customer is standing, you stand. If you want your customer to sit, ask the customer to sit first, and then you sit.
- Show interest in your customer's problem. Give your customer your undivided attention. Comments to a secretary or receptionist, such as, "Do not interrupt us," will show sincere interest.
- Physically display interest. Assume a body position, gestures, and tone of voice that show concern to your customer.
- React positively to your customer's concern. Show empathy for your customer's complaint. For example, if the customer indicates you have caused great inconvenience to their staff, apologize.

3.2.2.3.2 Step 2: Get the Facts

You cannot deal with your customer's problem until you know what it is. Do not deal with the symptoms; deal with the problem. An angry person is more likely to tell you symptoms than the real problem. To find out the facts, you need to do the following:

- Ask probing questions. A few examples are:

  "Give me an example of the problem.","Do you have any samples of defective products?", "Explain more about what you mean.", "Where did you get that piece of information?"

- Take detailed notes. Write down names, amounts in question, order numbers, dates or times at which events happened, specific products and parts of products where problems occurred.
- Observe feelings and attitudes. In many cases the problem may be more emotional than factual. However, you will need to deal with the emotions. Find out how a person feels about what has happened; find out what his colleagues or boss feels about the problem.
- Listen carefully to what is being said. Try to listen through the words to find out what the real problem is.

3.2.2.3.3 Step 3: Establish and Initiate an Action Program

Even if you believe that the complaint is not reasonable, you still need to take action. The action will serve two purposes: it will determine the validity of the facts and it will pacify the complainer. When taking action, you need to do the following:

- If an error has been made, admit it. Admit the error if you are responsible for the error. Do not minimize the seriousness of the error if it is serious. Not only admit it, but also apologize for it.
- Negotiate a satisfactory resolution with your customer. Only the customer knows what is satisfactory. Even if the solution is to conduct an investigation, it is an appropriate action if it is satisfactory to your customer.
- State the solution and get agreement from your customer. After you have agreed on what action to take, repeat it back to your customer and confirm agreement.
- Take action. Whatever you agreed to do, do it, and do it immediately. Just as it is important to begin communicating a solution within four minutes, it is equally important to resolve the action quickly.

3.2.2.3.4 Step 4: Follow Up with Your Customer

When the action you agreed upon has been taken, follow up with your customer to determine satisfaction. Just because you believe the problem has been solved, it is not logical to assume that your customer also agrees. The problem may be a difference of opinion about the solution. Words sometimes do not convey exactly what we mean. If your customer is happy with the resolution, the complaint has been finished. If your customer is not happy, more work has to be done, and you should go back to Step 2: Get the Facts, and try again.

3.2.3 Judgment

Judgment is a decision made by an individual. In sports the individual making the judgment is a referee, in law it is a judge. However, using judgment can apply to any activity. Judgment is normally a decision based on three criteria which are:

- Facts

  These are the indisputable evidence regarding a specific situation.

- Standards

  What the facts in the situation should be. Having the facts and the standards, the individual making the judgment at this point knows what has happened, and what should have happened.

- Experience

  An individual's involvement in similar situations, and using that experience to select the best decision which will have the minimum negative impact on the activity. However, if the standard is an objective standard, for example an athlete steps out of bounds then judgment reinforces the standard. However if the standard is subjective such as judging artistic impression in ice skating, then experience will play a much greater role in arriving at a decision.

3.2.4 Providing Constructive Criticism

In giving constructive criticism, you should incorporate the following tactics:

- Do it Privately

  Criticism should be given on a one-on-one basis. Only the individual being criticized should be aware that criticism is occurring. It is best done in a private location. Many times it is more effective if it is done in a neutral location, for example, in a conference room or while taking someone to lunch, rather than in the boss' office.

- Have the Facts

  General statements of undesired performance are not very helpful. For example, statements such as "That proposal is not clear, fix it" or "Your program does not make best use of the language or technology" leave people feeling confused and helpless. Before criticizing someone's performance, have specific items that are causing the deficiency or undesirable performance.

- Be Prepared to Help the Worker Improve His Performance

  It is not good enough to ask the worker to "fix it." You must be prepared to help fix it. Be prepared to train the subordinate in the area of deficiency. For example, in a proposal, indicate that a return-on-investment calculation was not made; or if a program failed to use the language properly, state specifically how it should and should not be used. You should not leave an individual feeling that they have performed poorly or unsure as to how to correct that performance.

- Be Specific on Expectations

  Be sure your subordinate knows exactly what you expect from him or her now and in the future. Your expectations should be as clear as possible so there can be no confusion. Again, in a proposal, indicate that you expect a return-on-investment calculation included in all proposals. Most people will try to do what they are expected to do—if they know what those expectations are.

- Follow a Specific Process in Giving Criticism

The specific process that is recommended is:

- State the positive first. Before criticizing indicate what you like about their performance. Again, be as specific as possible in the things you like.
- Indicate the deficiencies with products or services produced by the individual. Never criticize the individual, only the work performed by the individual. For example, never indicate that an individual is disorganized; indicate that a report is disorganized. People can accept criticism of their products and services; they have great difficulty when you attack their personal work ethic.
- Get agreement that there is a problem. The individual being criticized must agree there is a problem before proper corrective action can be taken. Avoid accepting agreement just because you are the boss; probe the need for improvement with the subordinate until you actually feel there is agreement that improvement can be achieved. For example, if you believe a report or program is disorganized, get agreement from the individual on specifically why it might be disorganized.
- Ask the subordinate for advice on how to improve their performance. Always try to get the employee to propose what needs to be done. If the employee's suggestion is consistent with what you have decided is a realistic method of improvement, you have finished the process.
- If the subordinate is unable to solve the problem, suggest the course of action that you had determined before performing the actual criticism.
- Make a specific "contract" regarding what will happen after the session. Be very specific in what you expect, when and where you expect it. If the employee is uncertain how to do it, the "contract" should include your participation, as a vehicle to ensure what will happen.
- One last recommendation for criticism:

  Avoid making threats about what will happen if the performance does not change. This will not cause any positive behavior change to occur and normally produces negative behavior. Leave the individual with the assumption that he or she has the capability for improvement, and that you know he or she will improve.

3.2.5 Project Relationships

The software tester is providing a service to those having a vested interest in the success of a software project. What is important about the relationship with the stakeholders are:

- The project relationships are defined

- The roles of each party and the relationships are defined
- The importance of the relationship to the success of the project is defined
- The influence that a party can have on software testing needs to be defined

An approach used by many organizations to document relationships is a project relationship chart illustrated in Figure 3-4.

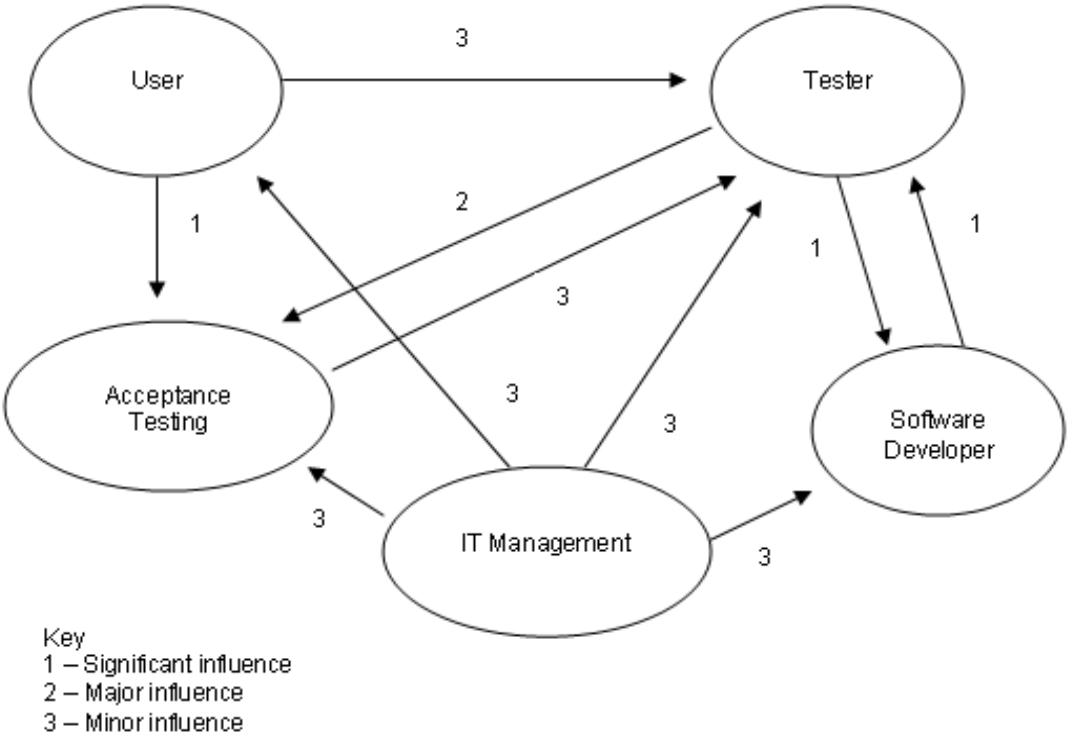


Figure 3-4 Project Relationship Chart

This chart is an example of a relationship chart and is constructed as follows:

1. Define the stakeholders

   All those individuals who have a stake in the success of the project, and thus must have a relationship to the project need to be defined. In our example, we defined the tester, the user, the software developer, IT management and acceptance testers.

2. Indicate the influence one stakeholder has on another stakeholder

   This chart uses a scale of “1”-to-“3” for influence:

   - 1 meaning a stakeholder has a significant influence over another stakeholder
   - 2 meaning a major influence
   - 3 meaning a minor influence

   Note that the lines show the direction of influence and the significance of the influence.

The purpose of this project relationship chart is to be assured that the tester has clearly identified which relationships are important to the success of software testing, and to be assured that the relationships will be developed and maintained during the software testing project.

3.2.6 Motivation, Mentoring, and Recognition

An important aspect of a supervisor's job is to motivate, mentor, and recognize the testers in his or her organization.

3.2.6.1 Motivation

Motivation has sometimes been defined as getting individuals to do work tasks they do not want to do or to perform those work tasks in a more efficient or effective manner.

Experience has shown that the motivation approach that works best is positive motivation. In other words don't attempt to motivate by fear or threats such as "no raises" or "termination."

Different people are motivated by different things. IBM at one time had a policy of supervisors asking their subordinates how they would like to be rewarded for a successful project. It is important to recognize that what motivates people is highly individualized.

The four most common motivators are:

1. Personal challenge – A job task which will challenge the individual's competency and capabilities.
2. Respect – Treating the individual as a professional.
3. Rewards – Some tangible thing that an individual will receive if they meet specific goals/ objectives.
4. Recognition – Publicizing among peers and management the value contributed by the individual.

3.2.6.2 Mentoring

Mentoring is helping or supporting an individual in a non-supervisory capacity. Mentors can be peers, subordinates, or superiors. What is important is that the mentor does not have a managerial relationship to perform the task of mentoring.

Mentoring can occur in any of the following three areas:

- Career counseling – Discussing career opportunities and assisting individuals in accomplishing their career objectives.
- Work tasks – Helping individuals achieve work tasks by either imparting the necessary skills or working with an individual in completing a job task.

- Professional advancement – Helping an individual achieve professional goals such as becoming a certified software tester (CSTE).

The only benefit a mentor receives for becoming a mentor is the satisfaction of helping another person succeed.

3.2.6.3 Recognition

Employees are recognized at the end of each pay period by being given a paycheck. However, motivation of employees can be increased by other recognition means. People like to be recognized for the contribution they make to a project.

The only key concept in this part of supervision is that recognition is important. However, recognition should not be a significant monetary value because obtaining that recognition may cause individuals to circumvent controls and good practices.

Some of the recognitions that have been used successfully within software testing are:

- Recognition by an IT manager at a formal IT meeting
- Group luncheons/group celebrations
- Tokens of appreciation such as a coupon for dinner out or a movie
- Time off if completing a task involved an excess of work hours
- Lunch with the boss

3.3 Test Leadership

All test managers are part manager and part leader. Most software test managers will spend most of their time managing and only a part of the time leading. However as testing moves into new areas such as testing to determine whether user success criteria have been achieved, the software test manager becomes more of a leader than a manager.

In discussing leadership, we will address these areas:

- Chairing meetings
- Team building
- Quality Management Organizational Structure
- Code of ethics

3.3.1 Chairing Meetings

Many IT staff members spend almost one half of their day in meetings. Meetings can be both productive and non-productive depending on how they are organized, run and meeting

decisions implemented. The software project manager in chairing meetings must be more of a leader than a manager.

The following guidelines on conducting meetings are common to most of the books and manuals on how to run an effective meeting. These guidelines are:

- Specific objectives to accomplish at the meeting must be defined
- Those having a stake in the potential decisions need to be represented at the meeting
- An agenda for the meeting, plus any background data, must be distributed to the attendees prior to the meeting allowing enough time for the individuals to prepare for the meeting discussions
- Rules for running the meeting need to be established such as Robert's Rules of Order
- The individual chairing the meeting must assure that all present have an equal opportunity to express their opinions
- A consensus process should be used to develop conclusions, actions to be taken, as a result of the meeting
- Specific responsibilities should be assigned to complete the actions
- Minutes of the meeting should be disseminated to the attendees within a reasonable period of time after the meeting concludes

3.3.2 Team Building

Much has been written about organization's loyalty to its employees and employee loyalty to organizations. R.M. Kanter stated that, "New loyalty is not to the boss or to the company, but to projects that actualize the mission and offer challenge, growth, and credit for results." What this tells the project leader is that team building needs to focus on the challenge, growth and credit an individual can achieve from working on a specific project.

This loyalty concept helps differentiate the challenge of the software project manager versus the traditional project manager. In projects where large portions of the project team are implementers rather than professionals, loyalty may be more to the supervisor or company. Implementers have different motivations and loyalties than do many IT professionals.

There are a myriad of books on team building. The objective of this discussion is not to duplicate what is available, but to focus on components of team building that are directed more at software teams, than traditional implementation teams. These components are: team development, team member interaction, team ethics, and team rewards.

3.3.2.1 Team Development

There are seven guidelines that are helpful in developing compatibility and motivation of a software project team:

1. Communicate the vision, objectives, and goals of the project.

   A software professional wants to know what the project is trying to accomplish. The vision indicates why the project is undertaken, the goals and objectives indicate what the project is to achieve. For example, the vision of a bank commercial loan software project might be to increase profitability. This specific objective might be to provide the loan officer the information needed to make a good loan decision.

2. Define roles and responsibilities of team members.

   Software projects, unlike non-software projects, have roles which are heavily people dependent and project scope dependent. It's important for professional staff to have those roles and responsibilities clearly defined. The staffing matrix described in an earlier part of this skill category would define those roles and responsibilities.

3. Empower team members to manage their responsibilities.

   Empowerment is a major motivator for professional people. Many of the agile concepts relate to empowerment. In other words, enable people to perform the tasks in the most efficient and effective manner. This helps eliminate barriers that increase costs and help project schedule.

4. Hold team members accountable for their assigned responsibilities in the team process.

   Team members need to have their work tasks well defined and then be held accountable for completing those work tasks. Managerial practices indicate that this process works best when individuals accept responsibility for performing tasks. Thus, having the Project Manager work individually with team members to assign team tasks they agree to perform, and then hold those individuals accountable for completing those tasks is an effective managerial practice.

5. Ensure that all the required skills are present on the team.

   Projects cannot be completed successfully if the team members lack the skills to complete the project. It is not necessary for every team member to have all the needed skills, but the team in total needs the skills. The staffing matrix helps assure that the appropriate skills exist within the project team.

6. Provide the necessary technical and team training.

   If the team lacks technical and team skills, the project manager should provide that training. Technical skills include the skills necessary to design and build the software, team skills to cover such skills as consensus building and conflict resolution.

7. Award successes and celebrate achievements.

   Establishing goals and objectives provides the basis for rewards and celebrations. While it's appropriate to reward and celebrate individual achievements, the team building necessitates team goals and team celebrations. These can be centered around milestones accomplished, as well as scoring high on customer satisfaction surveys.

3.3.2.2 Team Member Interaction

The following guidelines have proven effective in building an effective and cohesive team:

- Know communication and work preference styles of staff and assure that the team complements those communication and work preference styles.
- Set clear, measurable work requirement standards.
- Delegate authority to staff members that empowers them to perform the tasks in the manner they deem most effective and efficient.
- Exact responsibility and accountability for team members for completing their work tasks in an effective efficient manner with high quality work products.
- Give immediate and objective feedback to team members on the performance of their individual and team tasks.
- Communicate, communicate, communicate! Be certain that all team members know about any event that may impact team performance.

3.3.2.3 Team Ethics

The accounting and other corporate scandals during the past few years have undermined the public's confidence in corporations to act in an ethical manner, and to report truthful accounting data. In the U.S. these scandals resulted in the passage of the Sarbanes-Oxley Act which made unethical and improper accounting a criminal act and subjected the corporate management to jail sentences. Corporate ethics at all levels of an organization are important.

The following six attributes are associated with ethical team behavior:

1. Customer relations that are truthful and fair to all parties

   Ethical customer relations means treating customer/user personnel with integrity; not promising un-achievable results; and informing customer/users of problems that could have a negative impact on delivery or software performance; and striving to fully understand the user's true processing needs.

2. Protecting company property

   The team should not undertake any action that would have a negative impact on the protection of company property or subject that property to loss.

3. Compliance with company policies

   The team should be knowledgeable in company policies, considerate of those policies when making team decisions and taking only those actions that meet both the letter and intent of company policies.

4. Integrity of information

   Team should strive to ensure that information they produce is reliable and valid, and that the information is conveyed to the appropriate stakeholders on a timely basis.

5. Attendance

   Except for valid reasons, the team should be in attendance during normal work hours, be prompt for meetings, and during work hours devote their effort to performing work tasks.

6. Redefine standards of quality

The team should be knowledgeable in the quality standards for team deliverables, work in a manner that is conducive to meeting those quality standards, and if they cannot be met, inform the appropriate stakeholders of the lack of quality of team deliverables.

3.3.2.4 Team Rewards

Over the years, organizations have focused their reward system on individual performance. However, focusing only on individual rewards may undermine team cooperation and performance. If teams are the means by which work is performed, and software project development teams are the means for building software, then team rewards need to be incorporated into the organization's reward system.

There is no generally accepted approach for rewarding teams. The following reward systems have proven effective in organizations.

- Team celebrations

  At the conclusion of meeting team milestones and objectives, the team celebrates as a group at events such as luncheons, sporting activities, and other off-site events.

- Team financial rewards

  Teams are given a cash reward for meeting an objective or milestone and then the team splits the rewards amongst themselves in any manner in which the team determines appropriate.

- Team recognition

  Management recognizes the work of the team and recognizes that performance in such rewards as special parking spaces, lunch with the boss, time off with pay, and announcements in a variety of forms to the team's peers of the team's accomplishments.

3.3.3 Quality Management Organizational Structure

Until approximately 25 years ago almost all organizational structures were hierarchical. Direction came from the top down. The quality revolution significantly impacted the typical hierarchical structure. The structure was flattened, employees were empowered to make more decisions and new approaches to management were introduced.

The ability to become a leader is partially dependent upon whether the organization is a traditional hierarchical management approach, or the new quality management philosophy. The new quality management philosophy encourages leadership; the traditional hierarchical approach to management encourages managers.

Most managers practice traditional management. They have been taught to control their organization and employees, using an "I'll tell you what to do, and you'll do it" mentality. Many managers look at the short-term because their commitment to the organization is short range.

The key differences in philosophy between traditional management and quality management environments are illustrated in Table 3-3.

Table 3-3: Traditional versus Quality Management Philosophy

Traditional Management Philosophy	Quality Management Philosophy
Controls each result	Use the process
Who made the error?	What allowed the error?
Correct the error	Reduce variation and prevent the error
Employees are the problem	Refine the process
Management accountable to their manager	Management accountable to the customer
Competition between organizations	Teamwork
Motivation from fear of failure	Motivation from within (self)
Management of outputs (results)—focusing on detection of defects	Management of process inputs—methods or sources of variation that focus on preventing defects
Fire fighting	Continuous process improvement
Accomplishment from meeting quotas, the monthly or quarterly bottom line	Accomplishment from long-term impact of improving processes

The culture change required to build a quality management environment is significant. Management must change its philosophy, practices, and assumptions about work and people. The biggest mistake usually made when implementing a quality management environment is underestimating the cultural changes that must occur and the time required for accomplishing these changes. It is usually felt that only a few control charts are needed, and little effort is made to change the culture of the organization.

The programs needed to change from a traditional to quality management culture must be customized for an organization and its current culture. Table 3-4 illustrates cultural changes that can be made.

Table 3-4: Organizational Changes From Traditional Culture to a Quality Management Culture

Category	Traditional Culture	Quality Management Culture
Mission	Maximum return on investment (ROI), management by objectives (MBO)	Ethical behavior and customer satisfaction, climate for continuous improvement, ROI as a measure of performance
Customer Requirements	Incomplete or ambiguous understanding of customer requirements	Uses a systematic approach to seek out, understand and satisfy both internal and external customer requirements
Suppliers	Undirected relationship	Partnership
Objectives	Orientation to short-term objectives and actions with limited long-term perspective	Deliberate balance of long-term goals with successive short-term objectives
Improvement	Acceptance of process variability and subsequent corrective action as the norm	Understanding and continually improving the process
Problem Solving	Unstructured individualistic problem-solving and decision-making	Predominantly participative and interdisciplinary problem-solving and decision-making based on substantive data
Jobs and People	Functional, narrow scope, management controlled	Management and employee involvement, work teams, integrated functions
Management Style	Management style with uncertain objectives that instills fear of failure	Open style with clear and consistent objectives, encouraging group-derived continuous improvement
Role of Manager	Plan, organize, assign, control and enforce	Communicate, consult, delegate, coach, mentor, remove barriers, and establish trust
Rewards & Recognition	Pay by job, few team incentives	Individual and group recognition and rewards, negotiated criteria
Measurement	Orientation toward data gathering for problem identification	Data used to understand and continuously improve processes

3.3.4 Code of Ethics

Members of professional organizations have a responsibility to accept the standards of conduct that the organization represents. Those certified must maintain high standards of conduct in order to effectively discharge their responsibility.

3.3.4.1 Responsibility

Acceptance of any certification designation is a voluntary action. By acceptance, those certified assume an obligation of self-discipline beyond the requirements of laws and regulations.

The standards of conduct set forth in the code of ethics provide basic principles in the practice of IT quality assurance. Those certified should realize that their individual judgment is required in the application of these principles.

3.4 Managing Change

Once a test plan has been developed, it should be implemented. A test plan should be viewed as a contract between the testers and the stakeholders in the software system. If the test plan is executed as stated the test should be viewed as successful by the testers.

A test plan should also be viewed as a document that needs to be continually updated as circumstances change. When users make changes to the software system, the test plan needs to be changed. When circumstances surrounding the execution of the test plan changes, the test plan should change. For example, if the developers are late in getting the software to the testers, there may be time constraint problems. Thus the scope of testing may need to be changed because of the late delivery of the software components to the testers.

There are three test management activities that should be incorporated to ensure currentness and effectiveness of the test plan:

- Software Configuration Management
- Software Change Management
- Software Version Control

Each of these will be discussed individually.

3.4.1 Software Configuration Management

Software Configuration Management (CM) is a key component of the infrastructure for any software development organization. The ability to maintain control over the changes made to all project artifacts is critical to the success of a project. The more complex an application is, the more important it is to implement change to both the application and its supporting artifacts in a controlled manner.

Most organizations understand the importance of managing source code and the changes made to it, but all project artifacts need to be managed; from requirements and models, to test cases and test data. For example, if a requirement is changed or added during the project after the requirements have been baselined, the tests designed and built to validate the requirement

must also be updated. The appropriate version of the tests must be executed for the updated code, or incorrect results will be obtained.

For large projects, the Configuration Manager is often a full-time member of the project team. This person provides and supports the overall CM infrastructure and environment for the project team. Smaller efforts might not be able to support a full-time Configuration Manager, and might share a resource with other projects, or assign the role to someone on the project team in addition to their other responsibilities.

The CM function ensures that the CM environment facilitates product baselines, review, change, and defect tracking. Responsibilities also include writing the CM plan and reporting change request-based progress statistics. The CM function supports product development activities so that developers and integrators have appropriate workspaces to build and test their work, and that all artifacts are available as required.

The list below illustrates the types of project artifacts that must be managed and controlled in the CM environment:

- Source code
- Requirements
- Analysis models
- Design models
- Test cases and procedures
- Automated test scripts
- User documentation, including manuals and online Help
- Hardware and software configuration settings
- Other artifacts as needed

Multiple tools are available on the market to help teams manage project artifacts. Mainframe tools for source control like Change Man are typically used in conjunction with tools that support analysis and design artifacts created in a PC-based environment. A single tool, such as PVCS or Clear Case, can usually support client/server, Intranet, and Internet applications. If the application is implemented in an environment with multiple operating systems, e.g., UNIX and Windows NT, then the tool selected must support both environments.

3.4.2 Software Change Management

Managing software change is a process. The process is the primary responsibility of the software development staff. They must assure that the change requests are documented, that they are tracked through approval or rejection, and then incorporated into the developmental process. Many software development project teams group changes to be implemented into a new version of the system. This grouping normally reduces the costs associated with incorporating changes into the software requirements during development.

The testers need to know two aspects of change:

1. The characteristics of the change so that modification to the test plan and test data can be made to assure the right functionality and structure are tested.
2. The version in which that change will be implemented.

Without effective communication between the development team and the test team regarding changes, test effort may be wasted testing the wrong functionality and structure.

3.4.3 Software Version Control

Once dynamic testing begins, the project team must ensure that the appropriate versions of the software components are being tested. The time and effort devoted to testing are wasted if either the incorrect components, or the wrong version of the correct components, have been migrated to the test environment. The Configuration Manager must develop both migration and back-out procedures to support this process. These processes can be either manual or automated. Often, the CM tool selected will support the creation of automated build procedures that reduce the level of risk associated with migration.

3.4.3.1 Example

For discussion purposes, let's assume that you are working on a large, client/server, and iterative development project. The application architecture consists of a Windows NT Intranet Web Server, a Sun Solaris Application Server, a Sun Solaris Data Server, and a legacy application on the mainframe. The architecture alone has introduced a high level of risk into the project because of the multiple points of failure, and the multiple platforms and operating systems have added an additional amount of risk. To speed up development of the application, the project team has selected to employ an iterative development approach that will result in four iterative "builds" of the application. Each build is given a new release number, and integrates additional functionality with that delivered in the previous releases.

The development environment in place to support this project consists of the following:

- Four development servers, one for each of the development iterations.
- An Integration Test environment, where integration work is completed for each of the builds.
- A Test environment, where the test team conducts testing on each build.
- A Production environment, where users access and use the production version of the application.
- A Production Support environment, where the support team investigates reported problems and tests emergency fixes prior to moving them into production.

The first challenge for the CM function is to keep the environments configured appropriately. For example, the Production Support environment must be set up exactly as the Production environment, and must contain the same release version of the application components that

are running in Production. Emergency fixes must be applied to the version of the component running in production – not necessarily the latest version of the component, which might be vastly different from the production version if it is being enhanced for one of the future releases. If there is a problem with the new release migrated to production, the back-out procedures must be utilized to revert the environment back to the previous release of the software.

The Test environment configurations must also mirror the Production environment. This is critical if the test is to be successful at predicting the applications performance in the Production environment. Once the environment is configured correctly, the release that is ready for test must be migrated into the environment. Any defects that are identified can be corrected in the next “build” or release. If a release is either built incorrectly, or contains defects that prevent the continuation of testing, the back-out procedures can be utilized here as well.

Once a stable CM infrastructure is in place, the benefits of iterative development are easier to achieve. Without it, the development may be faster, but may also be more costly due to the management overhead associated with the build process.

This page intentionally left blank.

Skill Category

4

Test Planning

Testers need the skills to plan tests, including the selection of techniques and methods to be used to validate the product against its approved requirements and design. Test planning assesses the software application risks, and then develops a plan to determine if the software minimizes those risks. Testers must understand the development methods and environment to effectively plan for testing. The objective of this skill category is to explain the concept of risk, as well as the new and increased risks occurring within the computer environment. The tester must understand the new and increased risks in order to evaluate the controls in computer applications. Also explained is how you determine the magnitude of a risk so that it can be determined how many resources can be economically allocated to reduce that risk. The test plan should be focused on the identified risks.

Risk Concepts and Vocabulary	**4-1**
Risks Associated with Software Development	**4-4**
Risks Associated with Software Testing	**4-16**
Risk Analysis	**4-19**
Risk Management	**4-24**
Prerequisites to Test Planning	**4-27**
Create the Test Plan	**4-29**

4.1 Risk Concepts and Vocabulary

Understanding the definitions of the following terms will aid in comprehending the material in this category:

Test Case Test cases are how the testers validate that a software function, such as deducting a payroll tax, or a structural attribute of software, such as code complexity meets the software specifications (i.e., expected results).

Test Data Test data is information used to build a test case.

Test Script Test scripts are an online entry of test cases in which the sequence of entering test cases and the structure of the online entry system must be validated, in addition to the expected results from a single test case.

The following definitions are helpful when discussing all of the factors with risk and risk analysis.

Risk Risk is the potential loss to an organization, as for example, the risk resulting from the misuse of its computer. This may involve unauthorized disclosure, unauthorized modification, and/or loss of information resources, as well as the authorized but incorrect use of a computer. Risk can be measured by performing risk analysis.

Risk Analysis Risk analysis is an analysis of an organization's information resources, its existing controls, and its remaining organization and computer system vulnerabilities. It combines the loss potential for each resource or combination of resources with an estimated rate of occurrence to establish a potential level of damage in dollars or other assets.

Threat A threat is something capable of exploiting vulnerability in the security of a computer system or application. Threats include both hazards and events that can trigger flaws.

Vulnerability Vulnerability is a design, implementation, or operations flaw that may be exploited by a threat; the flaw causes the computer system or application to operate in a fashion different from its published specifications and to result in destruction or misuse of equipment or data.

Control Control is anything that tends to cause the reduction of risk. Control can accomplish this by reducing harmful effects or by reducing the frequency of occurrence.

The risk is turned into a loss by threat. A threat is the trigger that causes the risk to become a loss. For example, if fire is a risk, then a can of gasoline in the house, or young children playing with matches, are threats that can cause the fire to occur. While it is difficult to deal with risks, one can deal very specifically with threats.

Threats are reduced or eliminated by controls. Thus, control can be identified as anything that tends to cause the reduction of risk. In our fire situation, if we removed the can of gasoline

from the home or stopped the young children from playing with matches, we would have eliminated the threat and thus, reduced the probability that the risk of fire would be realized.

If our controls are inadequate to reduce the risk, we have vulnerability. Vulnerability, therefore, can be defined as a flaw in the system of control that will enable a threat to be exploited. For example, if our controls stated that no one should leave gasoline in the home, but did not inform our repair people of the control, they would produce a threat if they brought gasoline into the home. Thus, we would say that we had vulnerability in our control system.

The process of evaluating risks, threats, controls, and vulnerabilities is frequently called risk analysis. This is a task that the tester performs when he/she approaches the test planning from a risk perspective. Note that today many testers go through a formal risk analysis process because of the importance of correctly performing software processing tasks in a computerized business environment.

We will further explore the concepts of risk and risk analysis. The objective is to understand the importance of risk analysis in the test process. We will then review many of the new and increased risks associated with today's automated technology. This approach is designed to produce insight into the changing nature of the test process. As you review these risks, think how many of them might be possible in a manual environment, how many are unique to a computer environment, and which ones might be more severe in a computer environment.

Risk analysis attempts to identify all the risks and then quantify the severity of the risks. A risk is the potential for loss or damage to an organization from materialized threats. Risk can be measured to a large extent by performing a risk analysis process. A threat, as we have seen, is a possible damaging event. If it occurs, it exploits vulnerability in the security of a computer-based organization.

Risks, which are ever present in a computerized environment, are generated by a variety of threats. Some of these threats are physical – such as fire, water damage, earthquake, and hurricane. Other threats are people-oriented – such as errors, omissions, intentional acts to disrupt system integrity, and fraud. These risks cannot be eliminated, but controls can reduce the probability of the risk turning into a damaging event. A *damaging event* is the materialization of a risk to an organization's assets.

Testers should evaluate the software being tested to identify its vulnerability to materialization of risk. Vulnerability is a weakness that may be exploited by a threat to cause destruction or misuse of its assets or resources. In examining vulnerabilities, the tester also assesses the strength of the software controls that reduce the risk or vulnerability to an acceptable level.

The risks in a computerized environment include both the risks that would be present in manual processing and some risks that are unique or increased in a computerized environment. The tester must:

- Identify these risks
- Estimate the severity of the risk
- Develop tests to substantiate the impact of the risk on the application

For example, if the tester felt that erroneous processing was a very high risk for a specific application, then the tester should devise tests to substantiate the correctness or incorrectness of processing. The processing of a sample of transactions to determine whether or not the application had processed the transactions correctly could do this.

4.2 Risks Associated with Software Development

Each software system has a unique set of risks. Some of those risks are associated with the software functions, and other risks are associated with the process that develops the software. The risks associated with development should be assessed for each software system during development.

The risks associated with software development are listed below and then individually described:

- Improper use of technology
- Repetition of errors
- Cascading of errors
- Illogical processing
- Inability to translate user needs into technical requirements
- Inability to control technology
- Incorrect entry of data
- Concentration of data
- Inability to react quickly
- Inability to substantiate processing
- Concentration of responsibilities
- Erroneous or falsified input data
- Misuse by authorized end users
- Uncontrolled system access
- Ineffective security and privacy practices for the application
- Procedural errors during operations
- Program errors
- Operating system flaws
- Communications system failure

4.2.1 Improper Use of Technology

Computer technology provides systems analysts and programmers with a variety of processing capabilities. This technology must be matched to the needs of the user to optimize

the implementation of those needs. A mismatch of technology and needs can result in an unnecessary expenditure of organizational resources.

One of the more common misuses of technology is the introduction of new technology prior to the clear establishment of its need. For example, many organizations introduce web sites without clearly establishing the need for that technology. Experience has shown that the early users of a new technology frequently consume large amounts of resources during the process of learning how to use that technology.

Some of the types of conditions that lead to the improper use of technology are:

- Systems analysts and systems programmers improperly skilled in the use of technology
- Early user of new hardware technology
- Early user of new software technology
- Minimal planning for the installation of new hardware and software technology

4.2.2 Repetition of Errors

In a manual-processing environment, errors are made individually. Thus, a person might process one item correctly, make an error on the next, process the next twenty correctly, and then make another error. In automated systems, the rules are applied consistently. Thus, if the rules are correct, processing is always correct, but if the rules are erroneous, processing will always be erroneous.

Errors can result from application program problems, hardware failures, and failures in vendor-supplied software. For example, a wrong percentage may have been entered for FICA deductions; thus, every employee for that pay period will have the wrong amount deducted for FICA purposes.

The conditions that cause repetition of errors include:

- Inadequate checks on entry of master information
- Insufficient program testing
- Failure to monitor the results of processing

4.2.3 Cascading of Errors

The cascading of errors is the domino effect of errors throughout an application system. An error in one part of the program or application triggers a second (yet unrelated) error in another part of the application system. This second error may trigger a third error, and so on.

Cascading of errors is frequently associated with making changes to application systems. A change is made and tested in the program in which the change occurs. However, some

condition has been altered as a result of the change, which causes an error to occur in another part of the application system.

Cascading of errors can occur between applications. This risk intensifies as applications become more integrated. For example, a system that is accepting orders may be tied through a series of applications to a system that replenishes inventory based on orders. Thus, an insignificant error in the order-entry program can "cascade" through a series of applications – resulting in a very serious error in the inventory replenishment program.

The types of conditions that lead to cascading of errors include:

- Inadequately tested applications
- Failure to communicate the type and date of changes being implemented
- Limited testing of program changes

4.2.4 Illogical Processing

Illogical processing is the performance of an automated event that would be highly unlikely in a manual-processing environment. For example a payroll check being produced for a clerical individual for over $1 million. This is possible in an automated system as a result of programming or hardware errors, but highly unlikely in a manual system.

Computerized applications do not have human oversight comparable to that incorporated into manual systems. In addition, fewer people have a good understanding of the processing logic of computerized applications. Thus, in some instances illogical processing may not be readily recognizable.

The conditions that can result in illogical processing are:

- Failure to check for unusually large amounts on output documents
- Fields that are either too small or too large
- Failure to scan output documents

4.2.5 Inability to Translate User Needs into Technical Requirements

One of the major failures of system development has been a communication failure between users and project personnel. In many organizations, users cannot adequately express their needs in terms that facilitate the preparation of computerized applications. Likewise, the technical computer people are often unable to appreciate the concerns and requirements of their users.

This needs satisfaction risk is a complex risk. Exposures include failure to implement needs because users were unaware of technical capabilities; improperly implemented needs because the technical personnel did not understand user requirements; users accepting improperly

implemented needs because they are unsure how to specify changes; and the building of redundant manual systems to compensate for weaknesses in computerized applications.

The conditions that can lead to the inability to translate user needs into technical requirements include:

- Users without technical IT skills
- Technical people without sufficient understanding of user requirements
- Users' inability to specify requirements in sufficient detail
- Multi-user systems with no user "in charge" of the system

4.2.6 Inability to Control Technology

Controls are needed over the technological environment. The controls ensure that the proper version of the proper program is in production at the right time; that the proper files are mounted; that operators perform the proper instructions; that adequate procedures are developed to prevent, detect, and correct problems occurring in the operating environment; and that the proper data is maintained and retrievable when needed.

The types of conditions that result in uncontrolled technology include:

- Selection of vendor-offered hardware and software without consulting how to control them
- Too many control trade-offs for operational efficiency
- Inadequate restart and recovery procedures
- Inadequate control over different versions of programs

4.2.7 Incorrect Entry of Data

In computerized applications, there is a mechanical step required to convert input data into machine-readable format. In the process of conducting this task, errors can occur. Data that was properly prepared and authorized may be entered into computerized applications incorrectly. Much of the data entered today is entered by customers who are not employees. Some of the devices used by customers are unknown to the software developers. The data originator manually transcribes the input information and during this process, errors are made.

In the newer technology, data can be originated and entered at the same time. For example, scanners enter data directly into the computer system. However, errors can still occur during this process.

Newer methods of data entry include optical scanners and process control computers. The latter monitor such items as production machinery, automatic cash dispensers and point-of-sale equipment. These are all mechanical devices and thus, subject to failure.

The types of conditions that can cause incorrect entry of data include:

- Human errors in keying data
- Mechanical failure of hardware devices
- Misinterpretation of characters or meaning of manually recorded input
- Misunderstanding of data entry procedures

4.2.8 Concentration of Data

Computerized applications concentrate data in a format that is easy to access. In manual systems, data is voluminous and stored in many places. It is difficult for an unauthorized individual to spend much time undetected browsing through file cabinets or other manual storage areas.

But, using computerized media, unauthorized individuals can browse by using computer programs. This may be difficult to detect without adequate safeguards. In addition, the data can be copied quickly without leaving any visible trail or destroying the original data. Thus, the owners of the data may not be aware that the data has been compromised.

Database technology increases the risk of data manipulation and compromise. The more data stored in a single place, the greater the value of that information to an unauthorized individual. For example, the information about an individual in the payroll application is restricted to current pay information, but when that data is coupled with personnel history, not only is current pay information available, but also pay history, individual skills, years of employment, progression of employment and perhaps performance evaluations.

The concentration of data increases the problems of greater reliance on a single piece of data and reliance on a single computer file. If the fact entered is erroneous, the more applications that rely on that piece of data, the greater the impact of the error. In addition, the more applications that use the concentrated data, the greater the impact when the data is unavailable because of problems with hardware or software.

The types of conditions that can create problems attributable to the concentration of data in computerized applications include:

- Inadequate access controls enabling unauthorized access to data
- Erroneous data and their impact on multiple users of the data
- Impact of hardware and software failures that make the data available to multiple users

4.2.9 Inability to React Quickly

Much of the value of computerized applications is the ability to satisfy user needs on a timely basis. Some of these needs are predetermined and reports are prepared on a regular basis to meet these needs. Other needs occur periodically requiring special actions to satisfy them. If the computerized application is unable to satisfy these special needs on a timely basis, redundant systems may be built for that purpose.

One of the measures of success of a computerized application is the speed with which special requests can be satisfied. Some of the newer client-server applications with a query language can satisfy some requests within a very short time. On the other hand, some of the older batch-oriented applications may take several days or weeks to satisfy a special request. In some instances, the structuring of the application system is an inhibiting factor in satisfying requests. For example, if a user wanted all of the supporting information for a supply requisition in a batched system, the cost and difficulty of satisfying that request may be prohibitive. The reason is that the requisition could be spread over many weeks of processing – owing to back orders, returned shipments, and shipping errors. The evidence supporting the transaction may be spread over many different files and the cost of processing those files may be exorbitant.

The types of conditions that can cause computerized applications to be unable to react quickly include:

- The structure of the computer files is inconsistent with the information requested
- The general-purpose extract programs are not available to satisfy the desired request
- Computer time is unavailable to satisfy the request
- The cost of processing exceeds the value of the information requested

4.2.10 Inability to Substantiate Processing

Computerized applications should contain the capability to substantiate processing. This substantiation includes both the ability to reconstruct the processing of a single transaction and the ability to reconstruct control totals. Computerized applications should be able to produce all the source transactions that support a control total, and substantiate that any source document is contained in a control area.

Application systems need to substantiate processing for the purposes of correcting errors and proving the correctness of processing. When errors occur, computer personnel need to pinpoint the cause of those errors so they can be corrected. Computerized application customers, other users, and control-oriented personnel, such as auditors, frequently want to verify the correctness of processing.

The types of conditions that may result in the inability to substantiate processing include:

- Evidence is not retained long enough
- The cost of substantiating processing exceeds the benefits derived from the process
- The evidence from intermediate processing is not retained

4.2.11 Concentration of Responsibilities

The computerization of an application tends to concentrate the responsibilities of many people into the automated application. Responsibilities that had been segregated for control purposes among many people may be concentrated into a single application system. In addition, a

single application system may concentrate responsibilities from many departments within an organization.

The responsibilities in a computerized environment may be concentrated in both the application system and computer-oriented personnel. For example, the database administrator may absorb data control responsibilities from many areas in the organization. A single computer system project leader may have the processing responsibility for many areas in the organization. New methods of separation of duties must be substituted for the previous segregation of duties among people.

The types of conditions that cause the concentration of responsibilities in a computerized environment include:

- The establishment of a data processing programming and systems group to develop computerized applications for an organization.
- Establishment of large databases
- Client-server systems
- Web-based systems

4.2.12 Erroneous or Falsified Input Data

Erroneous or falsified input data is the simplest and most common cause of undesirable performance by an applications system. Vulnerabilities occur wherever data is collected, manually processed, or prepared for entry to the computer.

The types of conditions that cause erroneous or falsified input data include:

- Unreasonable or inconsistent source data values may not be detected
- Keying errors may not be detected
- Incomplete or poorly formatted transactions may be accepted and treated as if they were complete transactions
- Records in one format may be interpreted according to a different format
- An employee may fraudulently add, delete, or modify data (e.g., payment vouchers or claims) to obtain benefits (e.g., checks or negotiable coupons) for himself
- Lack of document counts and other controls over source data or input transactions may allow some of the data or transactions to be lost without detection – or allow extra records to be added
- Records about the personnel (e.g., a record of a personnel action) may be modified during entry
- Data that arrives at the last minute (or under some other special or emergency condition) may not be verified prior to processing
- Transactions in which errors have been detected may be corrected without verification of the full record

4.2.13 Misuse by Authorized End Users

End users are the people served by the IT system. The system is designed for their use, but they can also misuse it for undesirable purposes. It is often very difficult to determine whether their use of the system is in accordance with the legitimate performance of their job.

The types of conditions where misuse occurs by authorized end users are:

- An employee may convert information to an unauthorized use; for example, she or he may sell privileged data about an individual to a prospective employer, credit agency, insurance company, or competitor; or may use statistics for stock market transactions before their public release
- A user whose job requires access to individual records in a database may manage to compile a complete listing of the database and then make unauthorized use of it (e.g., sell a listing of employees' home addresses as a mailing list)
- Unauthorized altering of information may be accomplished by an authorized end user (e.g., theft of services)
- An authorized user may use the system for personal benefit (e.g., theft of services)
- A supervisor may manage to approve and enter a fraudulent transaction
- A disgruntled or terminated employee may destroy or modify records – possibly in such a way that backup records are also corrupted and useless
- An authorized user may accept a bribe to modify or obtain information.

4.2.14 Uncontrolled System Access

Organizations expose themselves to unnecessary risk if they fail to establish controls over who can enter the IT area, who can use the IT systems, and who can access the information contained in the system. Such uncontrolled access can lead to:

- Data or programs may be stolen from the computer room or other storage areas
- IT facilities may be destroyed or damaged by either intruders or employees
- Individuals may not be adequately identified before they are allowed to enter the IT area
- Unauthorized persons may not adequately protect remote terminals from use
- An unauthorized user may gain access to the system
- Passwords may be inadvertently revealed to unauthorized individuals. A user may write his/her password in some convenient place, or the password may be obtained from discarded printouts, or by observing the user as they type it.
- A user may leave a logged-in terminal unattended, allowing an unauthorized person to use it
- A terminated employee may retain access to an IT system because his name and password are not immediately deleted from authorization tables and control lists

- An unauthorized individual may gain access to the system for his own purposes (e.g., theft of computer services, or data, or programs, modification of data, alteration of programs, sabotage, and denial of services)
- Repeated attempts by the same user or terminal to gain unauthorized access to the system or to a file may go undetected

4.2.15 Ineffective Security and Privacy Practices for the Application

Inadequate manual checks and controls to ensure correct processing by the IT system or negligence by those responsible for carrying out these checks can result in many vulnerabilities:

- Poorly defined criteria for authorized access may result in employees not knowing what information they, or others, are permitted to access
- The person responsible for security or privacy may fail to restrict user access to only those processes and data that are needed to accomplish assigned tasks
- Large fund disbursements, unusual price changes, and unanticipated inventory usage may not be reviewed for correctness
- Repeated payments to the same party may go unnoticed because there is no review
- The application staff may carelessly handle sensitive data, by the mail service, or by other personnel within the organization
- Post-processing reports analyzing system operations may not be reviewed to detect security or privacy violations
- Inadvertent modification or destruction of files may occur when trainees are allowed to work on live data
- Appropriate action may not be pursued when a security variance is reported to the system security officer or to the perpetrating individual's supervisor; in fact, procedures covering such occurrences may not exist

4.2.16 Procedural Errors during Operations

Both errors and intentional acts committed by the operations staff may result in improper operational procedures and lapsed controls, as well as losses in storage media and output.

4.2.16.1 Procedures and Controls

- Files may be destroyed during database reorganization or during release of disk space
- Operators may ignore operational procedures – for example, allowing programmers to operate computer equipment
- Job control language parameters may be erroneous

- An installation manager may circumvent operational controls to obtain information
- Careless or incorrect restarting after shutdown may cause the state of a transaction update to be unknown
- Hardware maintenance may be performed while production data is online and the equipment undergoing maintenance is not isolated
- An operator may perform unauthorized acts for personal gain (e.g., make extra copies of competitive bidding reports, print copies of unemployment checks, and delete a record from journal file)
- Operations staff may sabotage the computer (e.g., drop pieces of metal into a terminal)
- The wrong version of a program may be executed
- A program may be executed twice using the same transactions
- An operator may bypass required controls
- Supervision of operations personnel may not be adequate during nighttime shifts
- Because of incorrectly learned procedures, an operator may alter or erase master files

4.2.16.2 Storage Media Handling

- Inadvertently or intentionally mislabeled storage media are erased. In a case where they contain backup files, the erasure may not be noticed until it is needed
- Backup files with missing or mislabeled expiration dates may be erased.
- Temporary files created during a job step, for use in subsequent steps, may be erroneously released or modified through inadequate protection of the files or because of an abnormal termination
- Storage media containing sensitive information may not get adequate protection because the operations staff is not advised of the nature of the information content
- Output may be sent to the wrong individual or terminal
- Improperly operating procedures in post processing tasks may result in loss of output
- Surplus output material (e.g., duplicates of output data, used carbon paper) may not be disposed of properly

4.2.17 Program Errors

Application programs should be developed in an environment that requires and supports complete, correct, and consistent program design, good programming practices, adequate testing, review, documentation, and proper maintenance procedures. Although programs developed in such an environment will still contain undetected errors, programs not developed in this manner will probably contain many errors. Additionally, programmers can deliberately modify programs to produce undesirable side effects, or they can misuse the programs they are in charge of:

- Records may be deleted from sensitive files without a guarantee that the deleted records can be reconstructed.

- Programmers may insert special provisions in programs that manipulate data concerning themselves (e.g., payroll programmer may alter own payroll records).
- Program changes may not be tested adequately before being used in a production run.
- Changes to a program may result in new errors because of unanticipated interactions between program modules.
- Program acceptance tests may fail to detect errors that occur for only unusual combinations of input (e.g., a program that is supposed to reject all except a specified range of values, actually accepts an additional value.)
- Programs, the content of which should be safeguarded, may not be identified and protected.
- Test data with associated output, or documentation for certified programs may not be retained for future use.
- Documentation for vital programs may not be safeguarded.
- Programmers may fail to keep a change log, or to maintain backup copies or to formalize record-keeping activities.
- An employee may steal programs he or she is maintaining and use them for personal gain (e.g., for sale to a commercial organization, or to hold another organization for extortion).
- Poor program design may result in a critical data value being initialized to zero. An error may occur when the program is modified to change a data value but only change it in one place.
- Production data may be disclosed or destroyed when used during testing.
- Errors may result when the programmer misunderstands requests for changes to the program.
- Programs may contain routines not compatible with their intended purpose, which can disable or bypass security protection mechanisms. For example, a programmer who anticipates being fired inserts a code into a program that will cause vital system files to be deleted as soon as his or her name no longer appears in the payroll file.
- Inadequate documentation or labeling may result in the wrong version of a program being modified.

4.2.18 Operating System Flaws

Design and implementation errors, system generation and maintenance problems, and deliberate penetrations resulting in modifications to the operating system can produce undesirable effects in the application system. Flaws in the operating system are often difficult to prevent and detect:

- User jobs may be permitted to read or write outside assigned storage area.
- Inconsistencies may be introduced into data because of simultaneous processing of the same file by two jobs.

- An operating system design or implementation error may allow a user to disable controls or to access all system information.
- An operating system may not protect a copy of information as thoroughly as it protects the original.
- Unauthorized modifications to the operating system may allow a data entry clerk to enter programs and thus subvert the system.
- An operating system crash may expose valuable information, such as password lists or authorization tables.
- Maintenance personnel may bypass security controls while performing maintenance work. At such times the system is vulnerable to errors or intentional acts of the maintenance personnel, or anyone else who might be on the system and discover the opening (e.g., micro coded sections of the operating system may be tampered with or sensitive information from online files may be disclosed).
- An operating system may fail to maintain an unbroken audit trail.
- When restarting after a system crash, the operating system may fail to ascertain that all terminal locations previously occupied are still occupied by the same individuals.
- A user may be able to get into a monitor or supervisory mode.
- The operating system may fail to erase all scratch space assigned to a job after the normal or abnormal termination of the job.
- Files may be allowed to be read or written prior to being opened by the operating system.

4.2.19 Communications System Failure

Information being routed from one location to another over communication lines is vulnerable to accidental failures and to intentional interception and modification by unauthorized parties.

4.2.19.1 Accidental Failures

- Undetected communications errors may result in incorrect or modified data.
- Information may be accidentally misdirected to the wrong terminal.
- Communication nodes may leave unprotected fragments of messages in memory during unanticipated interruptions in processing.
- Communication protocol may fail to positively identify the transmitter or receiver of a message.

4.2.19.2 Intentional Acts

- Unauthorized individuals may monitor communication lines
- Data or programs may be stolen
- Programs in the network that switch computers may be modified to compromise security

- Data may be deliberately changed by an individual's tapping the line (requires some sophistication, but is applicable to financial data).
- An unauthorized user may "take over" a computer communication port as an authorized user disconnects from it. Many systems cannot detect the change. This is particularly true in much of the currently available communication protocols.
- If encryption (i.e., use of codes) is used, keys may be stolen
- A user may be "spoofed" (i.e., tricked) into providing sensitive data
- False messages may be inserted into the system
- True messages may be deleted from the system
- Messages may be recorded and replayed into the system

4.3 Risks Associated with Software Testing

Testing is inherently a risk-based activity. Most companies would not pay to add testing to the cost of the project if there was not a cost associated with the risk of failure. Exhaustive testing is impractical for most applications under development. Exceptions include applications that support very high risk processes, such as air traffic control, nuclear power generation station operations, defense systems, and so on. The project team must design a test strategy that utilizes a balance of testing techniques to cover a representative sample of the system in order to minimize risk while still delivering the application to production in a timely manner.

It is the test manager's responsibility to determine how to apply the test methodology to achieve the greatest level of confidence in the application under development. Risk is a major driver in the test planning activity. When conducting risk analysis, two major components are taken into consideration:

- The probability that the negative event will occur
- The potential loss or impact associated with the event

The test manager must determine the appropriate amount of testing to perform based upon the risks associated with the application. These risks can arise from the newness and reliability of the technology being used, the nature of the application, or from the priority of the business functions under test. The amount of testing that should be performed is directly related to the amount of risk involved.

Understanding the risk-based nature of testing is also the key to dealing with the chronic problem of inadequate test resources. Risk must be used as the basis for allocating the test time that is available, and for helping to make the selection of what to test and how to allocate resources.

The test manager is also responsible for identification of potential risks that might impact testing. Some of the primary testing risks include:

- Not Enough Training/Lack of Test Competency

The majority of IT personnel have not been formally trained in testing, and only about half of full-time independent testing personnel have been trained in testing techniques. This causes a great deal of misunderstanding and misapplication of testing techniques.

- Us versus Them Mentality

  This common problem arises when developers and testers are on opposite sides of the testing issue. Each feels that it is “out to get” the other. Often, the political infighting takes up energy, sidetracks the project, and accomplishes little except to negatively impact relationships.

- Lack of Test Tools

  IT management may have the attitude that test tools are a luxury. Manual testing can be an overwhelming task. Trying to test effectively without tools is like trying to dig a trench with a spoon.

- Lack of Management Understanding and Support of Testing

  Support for testing must come from the top, otherwise staff will not take the job seriously and testers’ morale will suffer. Management support goes beyond financial provisions; management must also make the tough calls to deliver the software on time with defects or take a little longer and do the job right.

- Lack of Customer and User Involvement

  Users and customers may be shut out of the testing process, or perhaps they don’t want to be involved. Users and customers play one of the most critical roles in testing; making sure the software works from a business perspective.

- Not Enough Schedule or Budget for Testing

  This is a common complaint. The challenge is to prioritize the plan to test the right things in the given time.

- Over Reliance on Independent Testers

  Sometimes called the “throw it over the wall” syndrome, developers know that independent testers will check their work, so they focus on coding and let the testers do the testing. Unfortunately, this results in higher defect levels and longer testing times.

- Rapid Change

  In some technologies, especially Rapid Application Development (RAD), the software is created and modified faster than the testers can test it. This highlights the need for automation, but also for version and release management.

- Testers are in a Lose-Lose Situation

  On the one hand, if the testers report too many defects, they are blamed for delaying the project. Conversely, if the testers do not find the critical defects, they are blamed for poor quality.

- Having to Say “No”

Having to say, “No, the software is not ready for production,” is the single toughest dilemma for testers. Nobody on the project likes to hear that and frequently testers succumb to the pressures of schedule and cost.

- Test Environment

  The work environment is not conducive to effective and efficient testing.

- New technology

  The new focus on client/server, Intranet, and Internet applications has introduced even more risk to the test process. These multi-tiered systems are more complex than traditional mainframe systems, and therefore have a higher level of risk associated with their testing and implementation. Security, performance, availability, complex component integration, and a team new to the technology are just a few of these new risk factors.

- New developmental processes

  Along with these new delivery platforms come new methods for software development. Project teams have moved away from traditional “waterfall” methods, and follow a much more iterative approach to development and delivery. An integral part of this iterative approach is that testing activities are more integrated across the development cycle than before. Traditional system testing is not just conducted at the end of development, but may be conducted multiple times throughout the project for major components, and then completed once all development is complete.

Based on the risk assessment of the risks associated with the test process, changes to the test process may be needed. For example, if the testers lack training in the test process, that training should be provided prior to testing. If the training cannot occur then extra supervision would be required. If inadequate test resources are provided and the scheduled test time is not sufficient, the number of tests may need to be reduced.

Risk analysis will determine the risks associated with the software being tested and performing the test process. Risk analysis should determine the magnitude of the risks, and prioritize them in importance. Test planning will then consider those risks in developing the Test Plan.

4.3.1 Premature Release Risk

Premature release is defined as releasing the software into production under the following conditions:

- The requirements were implemented incorrectly
- The test plan has not been completed
- Defects uncovered in testing have not been corrected

Some IT organizations at a high maturity level have what is called a “defect expectation.” This means that testers expect a certain quantity of defects based on the history of software

developed using the software methodology. If testers fail to uncover the expected defect frequency they should be concerned about releasing the software.

The customer/user of software should expect operational problems if the software is released without completing the project or completing testing. These risks include the risks defined as software risks. The decision that the customer/user must make is, “Is the risk associated with premature release of software less than the risk of not releasing the software?”

In premature release of software the testers should include in their report the most probable outcomes associated with premature release. For example if they could not test requirements A, B and C, the test report should indicate that the functioning of those requirements has not been validated and the user should be alert to potential incorrect processing associated with those three requirements.

4.4 Risk Analysis

The objective of performing risk analysis as part of test planning is to help allocate limited test resources to those software components that pose the greatest risk to the organization. Testing is a process designed to minimize software risks. To make software testing most effective it is important to assure all the high risks associated with the software, will be tested first.

The decision the testers need to make is where to allocate test resources. Figure 4-1 provides an illustration of where to allocate test resources.

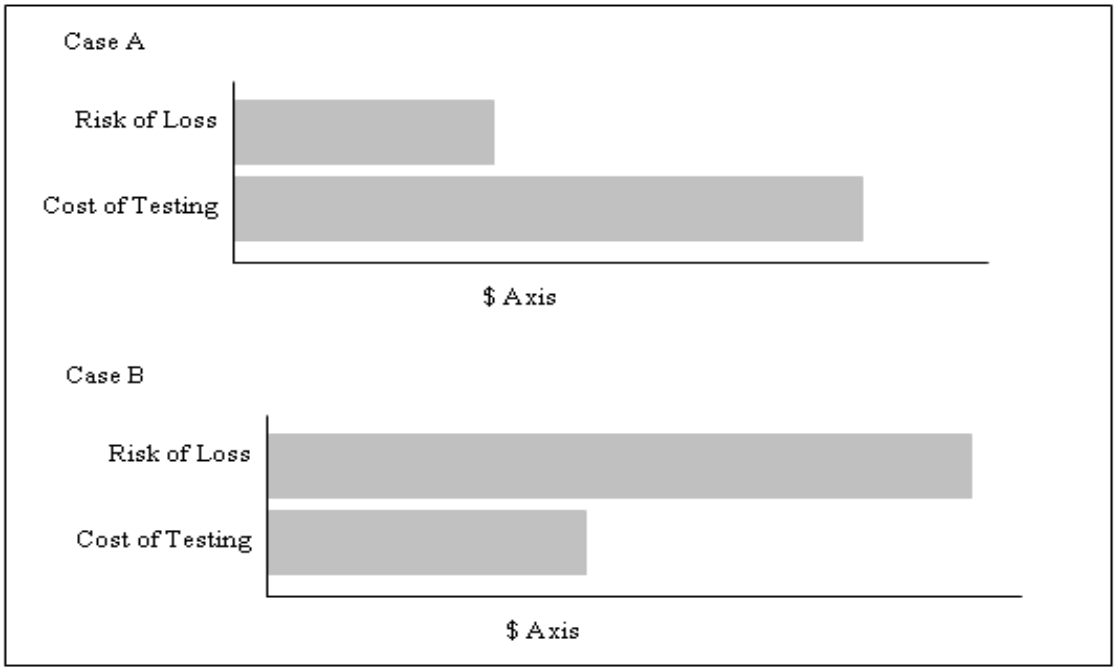


Figure 4-1 Illustrations of the Need for Software Testing

Case A shows that the risk of loss is much less than the cost of testing for that risk. For example, names on the master file may be misspelled but the impact of that defect is minimal, while testing to check that every name on a master file is correct could be expensive. On the other hand, case B shows that the cost of testing is much less than the potential loss. In that case, testing should occur. When the risk of loss and the cost of testing are close to the same dollar amount, a decision on how much testing should occur is a risk decision.

4.4.1 Risk Analysis Process

Performing risk analysis during test planning is a four-step process as follows:

1. Form the risk analysis team
2. Identify risks
3. Estimate the magnitude of the risk
4. Select testing priorities

4.4.1.1 Form the Risk Analysis Team

The key to a successful risk analysis is the establishment of the correct risk team, whose responsibility will be to identify risks and prioritize the use of test resources. The objective is to reduce the risks to an acceptable level.

The risk team may be part of the requirements team, or part of the test team, or it may be a team specifically selected for the purpose of completing risk analysis. The team should be comprised of three to six members and at a minimum possess the following skills:

- Knowledge of the user application
- Understanding of risk concepts
- Ability to identify controls
- Familiarity with both application and information services risks
- Understanding of information services concepts and systems design
- Understanding of computer operations procedures

The candidates included on the risk team should, at a minimum, include someone from the user area and any or all of the following:

- Software testing
- Risk consultant
- Project leader

4.4.1.2 Identify Risks

The objective of the risk team is first to identify the application-oriented, not environmental, risks associated with the application system. For example, the risks that relate to all

applications equally (i.e., environmental risks) need not be identified unless they have some special relevance to the applicants. The risk team can use one of the following two methods of risk identification:

- Risk analysis scenario

  In this method, the risk team "brainstorms" the potential application risks using their experience, judgment, and knowledge of the application area. It is important to have the synergistic effect of a group so that group members can challenge one another to develop a complete list of risks that are realistic for the application.

- Risk checklist

  The risk team is provided with a list of the more common risks that occur in automated applications. From this list, the team selects those risks that are applicable to the application. In this method, the team needs fewer skills because the risk list provides the stimuli for the process, and the objective of the team is to determine which of the risks on the list are applicable to the application.

4.4.1.3 Estimate the Magnitude of the Risk

The magnitude of a risk can be determined by any of the following means:

- Intuition and Judgment

  In this process, one or more individuals state they believe the risk is of a certain magnitude. The more respect awarded the individual proposing risk magnitude using this method, the greater the probability that it will be accepted. Often the risk will be categorized as high, medium or low.

- Consensus

  A team or group of people agrees to the severity of magnitude of a risk. The magnitude can be expressed in severity such as high, medium, or low, or an economic magnitude can be placed upon the risk. Normally, in instances where the loss associated with the risk is very high, it is generally not necessary to put a dollar value to the risk – the sheer magnitude of the risk makes the testing cost low in comparison.

- Risk Formula

  The risk formula can be used to calculate the magnitude of the risk. Using this process, the probability or frequency of an event occurring must be determined, and the loss per frequency must be determined. For example, if a type X risk occurs and the loss per occurrence is $500, then the loss associated with the risk is $500.

- Annual Loss Expectation (ALE) Estimation

  This process is a variation of the risk formula. However, while the risk formula usually requires the collection of operational data, the estimation process is a combination of consensus method and risk formula method. Also, it develops an annual estimate of loss.

  The premise behind the risk formula is that, in many cases, the benefits either greatly exceed the cost (see Figure 4-1 Illustrations of the Need for Software Testing on page 4-19) or the cost greatly exceeds the benefits. In these cases, estimations or approximations lead to an easy decision. However, if the range of estimated cost overlap

the range of estimated benefits, then additional investigation must be done. In the risk formula example above we would multiply the occurrences of loss per year, for example 100 times the loss ($500) to get an annual loss expectation of $50,000.

- A Process for Estimating the Annual Loss Expectation (ALE)

The following six tasks should be used to estimate ALE.

1. Make a preliminary assessment of the loss.

    - Estimate worst-case loss.
    - Use structured techniques for estimation of loss. Rather than select a specific dollar loss, we select a loss based on a multiple of 5, as shown in Figure 4-2. For example, if we thought our loss might be around $500 we would select the closest value of $625.

$ 1
$ 5
$ 25
$ 125
$ 625
$ 3,000
$ 16,000
$ 80,000
$ 400,000
$ 2,000,000
$ 10,000,000

Figure 4-2 Range of Values Based on Multiples of 5 Example

2. Make a preliminary assessment of frequency using a frequency table of multiples of five as in Figure 4-3.

625 times per day
125 times per day
25 times per day
5 times per day
1 time per day
Once in 5 days
Once in 1 month
Once in 4 months
Once in 2 years
Once in 10 years
Once in 50 years

Figure 4-3 Range of Frequencies of Occurrence Based on Multiples of 5 Example

3. Calculate an ALE using the loss and frequency tables in Figure 4-2 and Figure 4-3.
4. Narrow the ALE by:
   - Using multiple opinions to establish most likely ALE.
   - Rank intangible elements (see Figure 4-4 Preliminary Evaluation of Intangible Factors.)
   - Assign monetary values to intangibles.
   - Accumulate additional evidence, if needed.

Brief Description of Weakness	Not Important 0	1	2	3	Very Important 4	5
_____						
_____						
Cost Factors						
_____	☐	☐	☐	☐	☐	☐
_____	☐	☐	☐	☐	☐	☐
_____	☐	☐	☐	☐	☐	☐
Benefit Factors						
_____	☐	☐	☐	☐	☐	☐
_____	☐	☐	☐	☐	☐	☐
_____	☐	☐	☐	☐	☐	☐

Figure 4-4 Preliminary Evaluation of Intangible Factors

5. Make a post-analysis challenge of the decision.
6. Document the decision reached.

4.4.1.4 Select Testing Priorities

The risk team needs to rank the risks as an input to the test planning process. Risks are normally ranked by the severity of the risk. However, other considerations may impact the prioritization such as:

- Compliance required to laws and regulations
- Impact on competitiveness
- Impact on ethics, values and image

4.5 Risk Management

Risk management is a totality of activities that are used to minimize both the frequency and the impact associated with risks. The first part of risk management is to understand, identify

and determine the magnitude of risks. This was discussed in the previous section of this skill category.

The next component of risk management is determining risk appetite. Risk appetite defines the amount of loss management is willing to accept for a given risk. Risk appetite in a bank's credit card processing activity is the amount of losses they're willing to accept associated with credit card processing. Let us assume that the risk appetite is one percent of charges made to credit cards.

To manage a risk appetite of credit card losses of 1% the bank will first monitor to determine the actual losses. If the loss is less than 1% the bank may decide that they are losing potential business and issue many more credit cards. If the loss exceeds 1% the bank may reduce the credit limit on credit cards and/or terminate credit cards with individuals whose accounts are not current.

There are two activities associated with risk management which are:

- Risk Reduction Methods
- Contingency Planning

4.5.1 Risk Reduction Methods

The formula for quantifying risk also explains how to control or minimize risk. The formula to quantify risk is to multiply the frequency of an undesirable occurrence times the loss associated with that occurrence. For example, if ten customers a day terminated their business at a web site and the average customer places $50 orders, then the loss associated with this "unfriendly web site" would be 10 x $50, or $500.

Once the variables and the loss expectation formula have been defined, controls can then be identified to minimize that risk. This formula can also be used to calculate the cost/benefit of controls.

There are two means to reduce the frequency of occurrence of unfavorable events. As illustrated in Figure 4-5 How to Minimize Loss Due to Risk, these are:

- Reduce the opportunity for error
- Identify the error prior to loss.

Using a web site example, you could reduce the opportunity for customers to leave the web site by designing the web site so that any customer need can be identified in "three clicks" or less. To implement the other means, to identify the error prior to loss, you could track a customer as they move through the web site to determine if they are making too many clicks. If they make too many, transfer the customer to a web site operator who could correctly assist the customer.

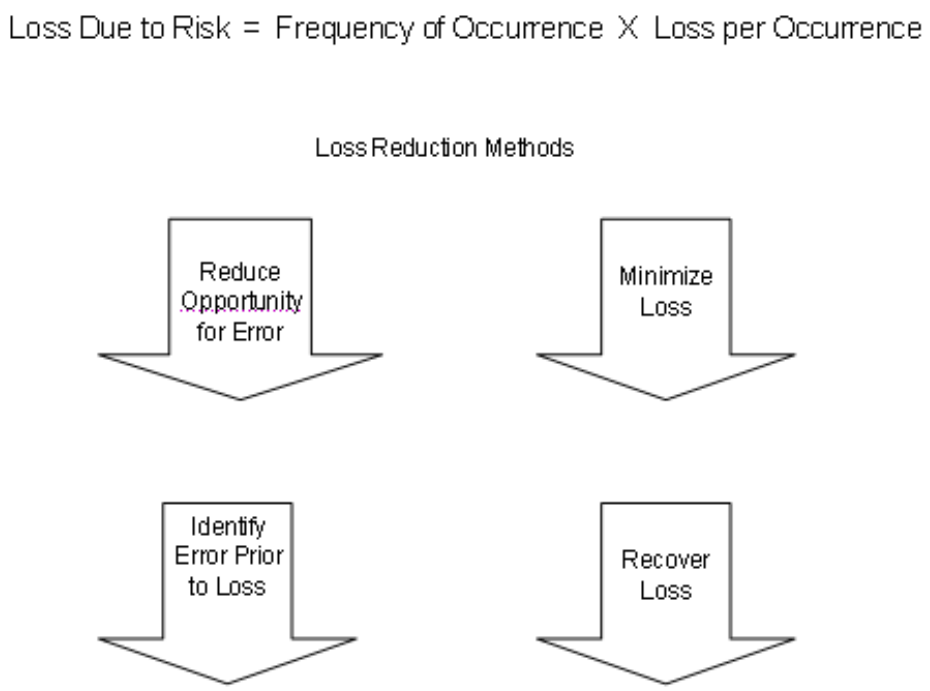


Figure 4-5 How to Minimize Loss Due to Risk

The loss per occurrence associated with an unfavorable event can likewise be reduced in two ways. The loss can be minimized, or an attempt can be made to recover the loss. Offering a customer the opportunity to talk directly to an individual might minimize the potential loss. If the customer's e-mail address can be captured, a follow-up message could go to the customer in an attempt to reestablish the relationship with that customer, and thus, recovering the loss of a sale.

When the estimated loss has been calculated, it can be compared against the cost of controlling that risk. When both the value of the loss has been estimated, and the cost of the controls has been estimated, the simple cost/benefit analysis can be performed to determine if it is economical to implement the controls. Obviously if the controls cost more than the estimated loss, the controls are not cost beneficial; on the other hand, if the controls are significantly less than the estimated loss, there is a good business case for building the controls.

4.5.2 Contingency Planning

Action plans should be established for activation when a loss is known to occur for a given risk. Using a non-IT example, organizations should predefine the actions to be taken should a fire occur. Actions of the plan would include evacuating the employees from the building.

All IT organizations should have plans for many contingencies. For example if computer operations terminate there is normally a plan to restart operations. If the computer site is lost due to fire or other causes there are often off-site data storage areas to enable processing to be reconstructed. For damage to the operation center there might be other facilities available for continuing operations.

What many IT organizations fail to do is develop contingency plans for the processing events occurring in the software system. For example if a computer is not operational, how are orders accepted. Stating that the computer is non-operational and that customers should call back is not an acceptable contingency plan.

The role of testers is to evaluate the adequacy of the contingency plans associated with risk. This should be a part of the test plan and the testing process.

4.6 Prerequisites to Test Planning

If test planning is viewed as a process, or a workbench, there are entrance criteria to the test planning process. The following entrance criteria are prerequisites to test planning:

- Test Objectives
- Acceptance Criteria
- Assumptions
- People Issues
- Constraints

4.6.1 Test Objectives

The test objectives include testing to assure that the software development project objectives are met; and testing to achieve the mission of the software testing group. Testing the functional and structural objectives of the software is accomplished to meet the quality definition category for “meeting requirements.” Testing to accomplish the needs of the users would fall in the quality definition category “fit for use.”

4.6.2 Acceptance Criteria

One of the major challenges the IT organization faces is communication. How can IT and the user of the software successfully communicate the characteristics of the desired software system? Many developmental methods are used to gather the software requirements such as joint application development. The purpose of these techniques is to facilitate communication between the IT organization and the users.

Another method of communicating is to have the user define the acceptance criteria for the software project. For example they might define the tasks that a data entry clerk must execute to process an incoming order. Testing that the functionality has been correctly installed would be included in the test plan. However the user might want to qualify that criterion stating that the functionality should be easy to use by data entry clerks with a certain pay grade or a certain level of education. In another example, the end user might specify that a certain level of accuracy is desired in order entry. You can assume this was a success or acceptance criteria by organizations like McDonald's when they converted their order entry equipment to a device that showed pictures of what could be ordered such as hamburgers, as opposed to having the order entry person enter the amount of the item.

4.6.3 Assumptions

In developing any type of plan certain assumptions exist. For example if a software system required a newly developed piece of hardware, an assumption could be that the hardware would be available on a specific date. The test plan would then be constructed based on that assumption. It is important that assumptions be documented for two reasons. The first is to assure that they are effectively incorporated into the test plan. The second is so that they can be monitored should the event included in the assumption not occur. For example hardware that was supposed to be available on a certain date, will not be available until three months later. This could significantly change the sequence and type of testing that occurs.

4.6.4 People Issues

People issues tend to be both political and personal. The people issues include: who should run the project, who can make decisions, and which organizational group has authority to decide requirements. Another possible issue is hearing "it's been tried before and didn't work." It is important to resolve these people issues prior to starting development.

Some organizations divide people into four categories when attempting to identify issues. These categories are:

- People who will make the software system happen
- People who will hope the software system happens
- People who will let the software system happen
- People who will attempt to make the software system not happen

If the stakeholders are divided among these four categories, issues are frequently apparent. For example, if two different business units want to make the software system happen, a decision would have to be made as to which would have primary responsibility and which would have secondary responsibility. If both want to have primary responsibility, conflict will occur.

4.6.5 Constraints

It has frequently been said that only “exhaustive testing” will uncover all defects. It is also said that exhaustive testing is neither practical nor economical. Thus without exhaustive testing there are constraints placed on the testers.

The obvious constraints are test staff size, test schedule and budget. Other constraints can include: inability to access user databases for test purposes; limited access to hardware facilities for test purposes; minimal user involvement in development of the test plan and testing activities.

Because constraints restrict the ability of testers to test effectively and efficiently, the constraints must be documented and integrated into the test plan. It is also important that the end users of the software understand the constraints placed on testers and how those constraints may impact the role and responsibility of software testers in testing their software system.

4.7 Create the Test Plan

Some insight into the importance of test planning:

> **“The act of designing tests is one of the most effective error prevention mechanisms known…**
>
> **The thought process that must take place to create useful tests can discover and eliminate problems at every stage of development.”**
>
> **Boris Beizer**

There is no one right way to plan tests. However, because of the importance of the test plan and the overall testing process the material in this section will provide tasks that are commonly followed to create a test plan. This section will also include “How-to” information in order to help understand the components of a software test plan. It is important to note that the material presented, while considered to be a “good practice,” is not the only way to create a test plan; other planning processes can be equally effective.

The test plan describes how testing will be accomplished. Its creation is essential to effective testing, and should take about one-third of the total test effort. If the plan is developed carefully, test execution, analysis, and reporting will flow smoothly. The time you spend will prove worthwhile with effective results.

The test plan should be an evolving document. As the developmental effort changes in scope, the test plan must change accordingly. It is important to keep the test plan current and to follow it. It is the execution of the test plan that management must rely on to assure that testing is effective. Also from this plan the testers ascertain the status of the test effort and base opinions on the results of the test effort.

The test plan should provide background information on the software being tested, test objectives and risks, and specific tests to be performed. Properly constructed, the test plan is a contract between the testers, and the project team and users describing the role of testing in the project. Thus, status reports and final reports will be based on that contract, which is the status of the planned test activities.

Test planning should begin at the same time requirements definition starts. The plan will be detailed in parallel with application requirements. During the analysis stage of the project, the test plan defines and communicates test requirements and the amount of testing needed so that accurate test estimates can be made and incorporated into the project plan.

The tests in the test plan should be repeatable, controllable, and ensure adequate test coverage when executed:

- Repeatable

  Once the necessary tests are documented, anyone on the test team should be able to execute the tests. If the test must be executed multiple times, the plan ensures that all of the critical elements are tested correctly. Parts or the entire plan can be executed for any necessary regression testing.

- Controllable

  Knowing what test data is required, when testing should be run, and what the expected results are all documented to control the testing process.

- Coverage

  Based on the risks and priorities associated with the elements of the system, the test plan is designed to ensure that adequate test coverage is built into the test. The plan can be reviewed by the appropriate parties to ensure that all are in agreement that the correct amount and types of tests are planned.

The following tasks are provided to help understand what is necessary to develop a good test plan:

- Understand the characteristics of the software being developed
- Build the test plan
- Write the test plan

4.7.1 Understand the Characteristics of the Software being Developed

Minimizing technological, business and test risks describe the broad objectives of testing. These are the risks and concerns that the testers need to be evaluating to assure the test objectives identified by the risks have been achieved. For example, if reliability is a risk for a specific project and up-time was a reliability measure, then the testers would need to evaluate the system's capability to meet that up-time objective.

The test team should investigate the project characteristics in order to evaluate the potential magnitude of the risk. During that investigation, the testers should at least do the following:

1. Define what it means to meet the project objectives.

   These are the objectives to be accomplished by the project team.

2. Understand the core business areas and processes.

   All information systems are not created equal. Systems that support mission-critical business processes are clearly more important than systems that support mission-support functions (usually administrative), although these, too, are necessary functions. Focusing on core business areas and processes is essential to the task of assessing the impact of the problem and for establishing the priorities for the program.

3. Assess the severity of potential failures.

   This must be done for each core business area and its associated processes.

4. Identify the components for the system.
   - Links to core business areas or processes
   - Platform languages, and database management systems
   - Operating system software and utilities
   - Telecommunications
   - Internal and external interfaces
   - Owners
   - Availability and adequacy of source code and associated documentation

5. Assure requirements are testable.

   Effective testing cannot occur if requirements cannot be tested to determine if requirements are implemented correctly.

6. Address implementation schedule issues.
   - Implementation checkpoints
   - Meetings
   - Identification and selection of conversion facilities
   - Time needed to put converted systems into production
   - The conversion of backup and archival data

7. Address interface and data exchange issues.

- Development of a model showing the internal and external dependency links among core business areas, processes, and information systems
- Notification of all outside data exchange entities
- Data bridges and filters
- Contingency plans if no data is received from an external source
- Validation process for incoming external data
- Contingency plans for invalid data

8. Evaluate contingency plans for this system and activities.

   These should be realistic contingency plans, including the development and activation of manual or contract procedures, to ensure the continuity of core business processes.

9. Identify vulnerable parts of the system and processes operating outside the information resource management area.

   Include telephone and network switching equipment and building infrastructure systems. Develop a separate plan for their testing.

4.7.2 Build the Test Plan

The development of an effective test plan involves the following tasks that are described below.

- Set test objectives
- Develop the text matrix
- Define test administration

4.7.2.1 Set Test Objectives

Test objectives need to be defined and agreed upon by the test team. These objectives must be measurable and the means for measuring defined. In addition, the objectives must be prioritized.

Test objectives should restate the project objectives from the project plan. In fact, the test plan objectives should determine whether those project plan objectives have been achieved. If the project plan does not have clearly stated objectives, then the testers must develop their own by:

- Setting objectives to minimize the project risks
- Brainstorming to identify project objectives
- Relating objectives to the testing policy, if established

The testers must have the objectives confirmed as the project objectives by the project team.

When defining test objectives, ten or fewer test objectives are a general guideline; too many distract the tester's focus. To define test objectives testers need to:

- Define each objective so that you can reference it by a number.
- Write the test objectives in a measurable statement, to focus testers on accomplishing the objective.
- Assign a priority to the objectives, such as:
  - High – The most important objectives to be accomplished during testing.
  - Average – Objectives to be accomplished only after the high-priority test objectives have been accomplished.
  - Low – The least important of the test objectives.

  Note: Establish priorities so that approximately one-third are High, one-third are Average, and one-third are Low.
- Define the acceptance criteria for each objective. This should state quantitatively how the testers would determine whether the objective has been accomplished. The more specific the criteria, the easier it will be for the testers to follow through.

During the test planning process there will be a decomposition of test objectives into test cases. At the conclusion of testing, the results of testing can be consolidated upward to determine whether or not the test objective has been accomplished.

4.7.2.2 Develop the Test Matrix

The test matrix is the key component of the test plan. It lists which software functions must be tested and the available tests. It shows “how” the software will be tested using checkmarks to indicate which tests are applicable to which functions. The test matrix is also a test “proof.” It proves that each testable function has at least one test, and that each test is designed to test a specific function.

An example of a test matrix is illustrated in Table 4-1. It shows four functions in a payroll system, with three tests to validate them. Since payroll is a batch system where data is entered all at one time, test data is also batched using various dates. The parallel test is run when posting to the general ledger, and all changes are verified through a code inspection.

Table 4-1: Payroll System Test Matrix Example

	Test Used to Test Function			
Software Function	**Desk Check**	**Parallel Test**	**Code Inspection**	**Validate Input**
Payroll Deduction Calculation	√		√	
Gross Pay	√		√	
Tax Deduction	√		√	
General Ledger Changes		√		

To develop a software test matrix, follow these steps. Each step is discussed below with examples.

- Define tests as required
- Define conceptual test cases to be entered as a test script
- Define verification tests
- Prepare the software test matrix

4.7.2.2.1 Define Tests as Required

A test case identified at the test plan level might "validate that all dating in a software function is correct." During execution, each date-related instruction in a software function would require a test case. (It is not necessary for test cases at the test planning level to be very detailed.)

Each test case should be named and numbered. Numbering is important both to control tests and to roll test results back to the high-level test described in the test plan.

Figure 4-6 shows an example of a hypothetical test to validate the inputs for a payroll application. Although all the detail may not be known because the data validation routines may not have been specified at this point, there is enough information to enable a group to prepare the data validation test cases. The test case(s) should validate that all invalid conditions will be rejected, and all valid conditions accepted.

Software Project: Payroll Application **Name of Test: Validate Input**
Test Objective Exercise data validation routines.
Test Input Prepare the following types of input data for each input field: • "Valid data • "Invalid data • "Range of codes • "Validation of legitimate values and tables
Test Procedures Create input transactions that contain the conditions described in test input. Run the entire test deck until all conditions are correctly processed.
Acceptance Criteria Software will reject all invalid conditions and accept all valid conditions.
Test Controls Run the entire test run each time the test is conducted. Rerun the test until all specified output criteria have been achieved.
Software or Structure Attribute Tested The data validation function

Figure 4-6 Simple Hypothetical Test for a Payroll Application

4.7.2.2.2 Define Conceptual Test Cases to be Entered as a Test Script

A conceptual test script is a high-level description of the test objectives, not the specific test cases that will be entered during online testing. From the test planning perspective, it is unimportant whether the individual items will be manually prepared, or generated and controlled using a software tool.

The example given for entering a batch test to validate date-related processing is also appropriate for test scripts. The primary differences are the sequence in which the events must occur and the source or location of the origin of the online event.

Table 4-2 shows an example of developing test scripts for the data-validation function of a data entry software project. It lists two scripting events, the evaluation criteria, and comments that would be helpful in developing these tests.

Table 4-2: Test Script Example for a Data Validation Function

Software Project: Order Entry **Software Module: Validate Input**				
Sequence	**Source**	**Script Event**	**Evaluation Criteria**	**Comments**
1	Data Entry Clerk	The data entry clerk enters an invalid customer order.	The customer number should be rejected as invalid.	A help routine would help to locate the proper customer number.
2	Data Entry Clerk	The data entry clerk enters a correct order into the system for one or more invalid company products.	The system should, first, confirm that the information entered is legal and for legitimate values, and second, ask the clerk to verify that all the information has been entered correctly.	This tests the entry of valid and invalid orders through the data validation routines.

4.7.2.2.3 Define Verification Tests

Verification is a static test performed on a document developed by the team responsible for creating software. Generally, for large complex documents, the verification process is a review; for smaller documents, the verification process comprises inspections. Other verification methods include:

- Static analyzers incorporated into the compilers
- Independent static analyzers
- Walkthroughs
- Confirmation in which a third-party attests to the accuracy of the document

Verification tests normally relate to a specific software project, but because of the extensiveness of testing, a single verification test may be applicable to many software projects. For example, it may be determined that each source code listing that is changed will be inspected prior to the unit test.

4.7.2.2.4 Prepare the Software Test Matrix

The objective of this matrix, as shown in Table 4-3, is to illustrate how to document which test cases test which software function and which structural attribute.

Table 4-3: Software Test Matrix Example

	Software Function / Structural Attribute									
Test Cases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Test A										
Test B										
Test C										
Test D										
Test E										
Test F										

The vertical axis of the matrix lists the software function or software attributes of the system. The horizontal axis lists the test cases to be conducted on those software attributes. The intersection of the vertical and horizontal axes indicates expected test results. A check mark can be used to indicate the test showed the correct results.

4.7.2.3 Define Test Administration

The administrative component of the test plan identifies the schedule, milestones, and resources needed to execute the test plan as illustrated in Table 4-4, “Test Administrative Worksheet Example” and a representative worksheet in Figure 4-7 Administrative Worksheet for an Administrative Checkpoint. This cannot be undertaken until the technical part, that is, the test matrix has been completed.

Prior to developing the test plan, the test team has to be organized. This initial test team is responsible for developing the test plan and then defining the administrative resources needed to complete the plan. Thus, part of the plan will be executed as the plan is being developed; that part is the creation of the test plan, which itself consumes resources.

The test plan, like the implementation plan, is a dynamic document, meaning it changes as the implementation plan changes and the test plan is being executed. The test plan must be viewed as a “contract” in which any modifications must be incorporated.

The following is the type of information normally included for each administrative checkpoint in the test plan.

Table 4-4: Test Administrative Worksheet Example

Software Project	The name or number that uniquely identifies the project or system that will be tested for compliance.
Project	The name of the project being tested.
Checkpoint for Administration	The name of the systems development checkpoint documents. If the checkpoint document has not been completed, the name of the checkpoint. Testing can be extremely difficult to perform at this checkpoint.
Schedule	The dates on which the following items need to be started and completed: Plan Train test group Obtain data Test execution Test Report(s)
Budget	The test resources allocated at this milestone, including both execution and test analysis and reporting.
Resources	The resources needed for this checkpoint, including: Equipment (computers and other hardware for testing) Software and test personnel (staff to be involved in this milestone test, designated by name or job function)
Testing Materials	Materials needed by the test team to perform the test at this checkpoint, including: System documentation (specific products and documents needed to perform the test at this point) Software to be tested (names of the programs and subsystems to be tested at this point) Test input (files or data used for test purposes) Test documentation (any test documents needed to conduct a test at this point) Test tools (software or test tools needed to conduct the test at this point)
Test Training	It is essential that the test team be taught how to perform training. They may need specific training in the use of test tools and test materials, the performance of specific tests, and the analysis of test results.

Software Project:			
Test Milestone Number:			
		Start	Finish
Schedule:			
Tester Training:			
Test Cases:			
Execution:			
Report:			
Budget:			
Resources			
Equipment:			
Support Personnel:			
Test Personnel:			
Testing Materials			
Project Documentation:			
Software to be Tested:			
Test Cases:			
Test Documentation:			
Test Tools:			
Test Training			

Figure 4-7 Administrative Worksheet for an Administrative Checkpoint

4.7.2.4 State Test Plan General Information

The general information is designed to provide background and reference data on testing. Because of the scope of this testing, in many organizations this background information will be necessary to acquaint testers with the project.

The general information about the test project normally includes the following information:

- Software Project

  The name or number that uniquely identifies the project or system that will be tested for compliance.

- Summary

  A one or two paragraph overview of what is to be tested and how the testing will be performed.

- Pretest Background

Summary of any previous test experiences that might prove helpful with testing. The assumption is, if there were problems in the past, they will probably continue; however, if there were a few problems with test tools, the test team can expect to use these tools effectively.

- Test Environment

  The computer center or facilities used to test the application. In a single computer center installation, this subsection is minimal. If the software is used in multiple installations, the test environments may need to be described extensively.

- Test Constraints

  Certain types of testing may not be practical or possible during testing. For example, in banking systems in which the software ties to the Fed Wire system, it is not possible to test software with that facility. In other cases, the software cannot yet interface directly with production databases, and therefore the test cannot provide assurance that some of these interfaces work. List all known constraints.

- References

  Any documents, policies, procedures, or regulations applicable to the software being tested or the test procedures. It is also advisable to provide a brief description of why the reference is being given and how it might be used during the testing process.

- When to Stop

  What type of test results or events should cause testing to be stopped and the software returned to the implementation team for more work.

4.7.2.4.1 Define Test Milestones

Test milestones are designed to indicate the start and completion date of each test. These tests are derived from the test plan. The start and completion milestones should be listed as numbers. If you prefer, these may be days or dates. For example, milestone 1 could just be week 1 or day 1, or November 18. The tests from the test matrix are then listed.

Note that organizations that have scheduling software should include the person responsible for performing that test, as the assignment becomes known.

4.7.3 Write the Test Plan

The test plan may be as formal or informal a document as the organization's culture dictates. When a test team has completed testing, they have basically completed the test plan. If a formal test plan is required, the information from the various forms that were completed above can be used to complete a plan as shown in Figure 4-8. If an informal test plan is required, the various forms that were completed above are adequate. Generally, if the test team is small, the completed forms are more than adequate. As the test team grows in size, it is generally better to formalize the test plan.

4.7.3.1 Guidelines to Writing the Test Plan

Test planning can be one of the most challenging aspects of testing. The following guidelines can help make the job a little easier.

- Start early

  Even though you might not have all of the details at hand, you can complete a great deal of the planning effort by starting on the general and working toward the specific. By starting early, you can also identify resource needs and plan for them before other areas of the project subsume them.

- Keep the Test Plan flexible

  Make it easy to add test cases, test data, and so on. The test plan itself should be changeable, but subject to change control.

- Review the Test Plan frequently

  Other people's observations and input greatly facilitate achieving a comprehensive test plan. The test plan should be subject to quality control just like any other project deliverable.

- Keep the Test Plan concise and readable

  The test plan does not need to be large and complicated. In fact, the more concise and readable it is, the more useful it will be. Remember, the test plan is intended to be a communication document. The details should be kept in a separate reference document.

- Calculate the planning effort

  You can count on roughly one-third of the testing effort being spent on each of the following test activities: planning, execution, and evaluation.

- Spend the time to do a complete Test Plan

  The better the test plan, the easier it will be to execute the tests.

4.7.3.2 Test Plan Standard

There is no one universally accepted standard for test planning. However, there is great consistency between the different organizations that have defined a test plan standard. This section will begin with a discussion of what is normally contained in a test plan, and then provide an example of a test plan standard that is consistent with the test plan standards provided by major standard-setting bodies such as the Institute of Electrical and Electronics Engineers (IEEE) and the National Institute of Standards in Technology (NIST) a part of the U.S. government.

Test Plans and their formats vary from company to company, but the best examples contain most of the elements discussed here. The Table of Contents of a test plan might contain the following:

- Test Scope

- Test Objectives
- Assumptions
- Risk Analysis
- Test Design
- Roles & Responsibilities
- Test Schedule & Resources
- Test Data Management
- Test Environment
- Communication Approach
- Test Tools

4.7.3.2.1 Test Scope

- This section answers two equally important questions: “What will be covered in the test?” and “What will not be covered in the test?” The answers to either of these questions might include:
- Specific functional or structural requirements
- System interfaces
- Infrastructure components (e.g., network stability)
- Supplemental deliverables, such as application documentation

4.7.3.2.2 Test Objectives

A test objective is simply a testing “goal.” It is a statement of what the tester is expected to accomplish or validate during a specific testing activity. Test objectives:

- Guide the development of test cases, procedures, and test data.
- Enable the tester and project managers to gauge testing progress and success.
- Enhance communication both within and outside of the project team by helping to define the scope of the testing effort.

Each objective should include a high-level description of the expected test results in measurable terms, and should be prioritized. In cases where test time is cut short, test cases supporting the highest priority objectives would be executed first.

4.7.3.2.3 Assumptions

These assumptions document test prerequisites, which if not met, could have a negative impact on the test. The test plan should communicate the risk that is introduced if these expectations are not met. Examples of assumptions include:

- Skill level of test resources
- Test budget
- State of the application at the start of testing
- Tools available

- Availability of test equipment

Entrance and exit criteria for each stage of testing could be documented here.

4.7.3.2.4 Risk Analysis

Although the test manager should work with the project team to identify risks to the project, this section of the plan documents test risks and their possible impact on the test effort. Some teams may incorporate these risks into project risk documentation if available. Risks that could impact testing include:

- Availability of downstream application test resources to perform system integration or regression testing
- Implementation of new test automation tools
- Sequence and increments of code delivery
- New technology

4.7.3.2.5 Test Design

The test design details the following:

- The types of tests that must be conducted
- The stages of testing that are required (e.g., Unit, Integration, System, Performance, and Usability)
- Outlines the sequence and timing of tests

4.7.3.2.6 Roles & Responsibilities

This section of the test plan defines who is responsible for each stage or type of testing. A responsibility matrix is an effective means of documenting these assignments. Note that although the Test Manager usually writes the test plan, it does not just include information on tests that the test team will execute.

4.7.3.2.7 Test Schedule & Planned Resources

The test schedule section includes the following:

- Major test activities
- Sequence of tests
- Dependence on other project activities
- Initial estimates for each activity

The plan should not be maintained separately, but incorporated into the overall Project Plan. Test resource planning includes:

- People, tools, and facilities
- An analysis of skill sets so that training requirements can be identified

4.7.3.2.8 Test Data Management

This section of the plan defines the data required for testing, as well as the infrastructure requirements to manage test data. It includes:

- Methods for preparing test data
- Backup and rollback procedures
- High-level data requirements, data sources, and methods for preparation (production extract or test data generation)
- Whether data conditioning or conversion will be required
- Data security issues

4.7.3.2.9 Test Environment

Environment requirements for each stage and type of testing should be outlined in this section of the plan, for example:

- Unit testing may be conducted in the development environment, while separate environments may be needed for integration and system testing
- Procedures for configuration management, release, and version control should be outlined
- Requirements for hardware and software configurations
- The location of individual test events
- The defect tracking mechanisms to be used

4.7.3.2.10 Communication Approach

In the complex, matrix environment required for testing in most companies, various communication mechanisms are required. These avenues should include

- Formal and informal meetings
- Working sessions
- Processes, such as defect tracking
- Tools, such as issue and defect tracking, electronic bulletin boards, notes databases, and Intranet sites
- Techniques, such as escalation procedures or the use of white boards for posting current state of testing (e.g., test environment down)
- Miscellaneous items such as project contact lists, meeting audiences, and frequency of defect reporting

4.7.3.2.11 Tools

Any tools that will be needed to support the testing process should be included here. Tools are usually used for:

- Workplan development
- Test planning and management

- Configuration management
- Test script development
- Test data conditioning
- Test execution
- Automated test tools
- Stress/load testing
- Results verification
- Defect tracking

The information outlined here cannot usually all be completed at once, but is captured in greater levels of detail as the project progresses through the life cycle.

A four-part test plan standard is provided in Figure 4-8 and Figure 4-9 as an example.

1. GENERAL INFORMATION
   - 1.1 Summary / Scope
     Summarize the functions of the software and the tests to be performed.
   - 1.2 Environment and Pretest Background
     Summarize the history of the project. Identify the user organization and computer center where the testing will be performed. Describe any prior testing and note results that may affect this testing.
   - 1.3 Test Objectives
     State the objectives to be accomplished by testing.
   - 1.4 Expected Defect Rates
     State the estimated number of defects for software of this type.
   - 1.5 References
     List applicable references, such as:
     - a) Project request authorization.
     - b) Previously published documents on the project.
     - c) Documentation concerning related projects.

2. PLAN
   - 2.1 Software Description
     Provide a chart and briefly describe the inputs, outputs, and functions of the software being tested as a frame of reference for the test descriptions.
   - 2.2 Test Team
     State who is on the test team and their test assignment(s).
   - 2.3 Milestones
     List the locations, milestone events, and dates for the testing.
   - 2.4 Budgets
     List the funds allocated to test by task and checkpoint.
   - 2.5 Testing (Systems Checkpoint)
     Identify the participating organizations and the system checkpoint where the software will be tested.
     - 2.5.1 Schedule (and budget)
       Show the detailed schedule of dates and events for the testing at this location. Such events may include familiarization, training, data, as well as the volume and frequency of the input. Resources allocated for tests should be shown.
     - 2.5.2 Requirements
       State the resource requirements, including:
       - a. Equipment
         List the expected period of use, types, and quantities of the equipment needed.
       - b. Software
         List other software that will be needed to support the testing that is not part of the software to be tested.
       - c. Personnel
         List the numbers and skill types of personnel that are expected to be available during the test from both the user and development groups. Include any special requirements such as multi-shift operation or key personnel.
     - 2.5.3 Testing Materials
       List the materials needed for the test, such as:
       - a. System documentation
       - b. Software to be tested and its medium
       - c. Test Inputs
       - d. Test documentation
       - e. Test Tools

Figure 4-8 Example of a System test Plan

2.5.4 Test Training
Describe or reference the plan for providing training in the use of the software being tested. Specify the types of training, personnel to be trained, and the training staff.

2.5.5 Test to be Conducted
Reference specific tests to be conducted at this checkpoint.

2.6 Testing (System Checkpoint)
Describe the plan for the second and subsequent system checkpoints where the software will be tested in a manner similar to paragraph 2.5.

3. SPECIFICATIONS AND EVALUATION

3.1 Specifications

3.1.1 Business Functions
List the business functional requirements established by earlier documentation, or Task 1 of Step 2.

3.1.2 Structural Functions
List the detailed structural functions to be exercised during the overall test.

3.1.3 Test/Function Relationships
List the tests to be performed on the software and relate them to the functions in paragraph 3.1.2.

3.1.4 Test Progression
Describe the manner in which progression is made from one test to another so that the entire test cycle is completed.

3.2 Methods and Constraints

3.2.1 Methodology
Describe the general method or strategy of the testing.

3.2.2 Test Tools
Specify the type of test tools to be used.

3.2.3 Extent
Indicate the extent of the testing, such as total or partial. Include any rationale for partial testing.

3.2.4 Data Recording
Discuss the method to be used for recording the test results and other information about the testing.

3.2.5 Constraints
Indicate anticipated limitations on the test due to test conditions, such as interfaces, equipment, personnel, or databases.

3.3 Evaluation

3.3.1 Criteria
Describe the rules to be used to evaluate test results, such as range of data values used, combinations of input types used, and maximum number of allowable interrupts or halts.

3.3.2 Data Reduction
Describe the techniques to be used for manipulating the test data into a form suitable for evaluation, such as manual or automated methods, to allow comparison of the results that should be produced to those that are produced.

4. TEST DESCRIPTIONS

4.1 Test (Identify)
Describe the test to be performed (format will vary for online test script).

4.1.1 Control
Describe the test control, such as manual, semiautomatic, or automatic insertion of inputs, sequencing of operations, and recording of results.

4.1.2 Inputs
Describe the input data and input commands used during the test.

4.1.3 Outputs
Describe the output data expected as a result of the test and any intermediate messages that may be produced.

4.1.4 Procedures
Specify the step-by-step procedures to accomplish the test. Include test setup, initialization, steps and termination.

4.2 Test (Identify)
Describe the second and subsequent tests in a manner similar to that used in paragraph 4.1.

Figure 4-9 Example of a System test Plan (continued)

This page intentionally left blank.

Skill Category

5

Executing the Test Plan

The test plan should be executed as designed. If the plan cannot be executed as designed it should be changed, or notations made as to what aspects of the plan were not performed. Testing according to the test plan should commence when the project commences and conclude when the software is no longer in operation. Portions of the test plan can be performed while the test plan is being written. To carry out the test plan, testers require many skills including designing test cases and test scripts, proper and efficient use of test tools, execute tests, recording test results, and managing defects.

Test Case Design	**5-1**
Test Coverage	**5-26**
Performing Tests	**5-27**
Recording Test Results	**5-32**
Defect Management	**5-37**

5.1 Test Case Design

The test objectives established in the test plan should be decomposed into individual test cases and test scripts. In Skill Category 4 the test plan included a test matrix that correlates a specific software function to the tests that will be executed to validate that the software function works as specified.

To create this test matrix, begin the process with high-level test objectives. These are decomposed into lower and lower objectives until functional and structural test objectives are defined individually. At that point, test cases and scripts can be prepared to validate those individual structural and functional conditions.

When the objectives have been decomposed, or decomposed to a level that the test case can be developed, a set of conditions can be created which will not only test the software during development, but can test changes during the operational state of the software.

You can design and prepare the following types of test cases:

- Functional
- Structural
- Erroneous
- Stress
- Scripts
- Use Cases

5.1.1 Functional Test Cases

Functional analysis seeks to verify, without execution, that the code faithfully implements the specification. Various approaches are possible. In proof of correctness, a formal proof is constructed to verify that a program correctly implements its intended function. In safety analysis, potentially dangerous behavior is identified and steps are taken to ensure such behavior is never manifested. Functional analysis is mentioned here for completeness, but a discussion of it is outside the scope of this section.

5.1.1.1 Design Specific Tests for Testing Code

Program testing is functional when test data is developed from documents that specify a module's intended behavior. These documents include, but are not limited to, the actual specification and the high-and-low-level design of the code to be tested. The goal is to test the specified behavior for each software feature, including the input and output.

5.1.1.2 Functional Testing Independent of the Specification Technique

Specifications detail the assumptions that may be made about a given software unit. They must describe the interface through which access to the unit is given, as well as the behavior once such access is given. The interface of a unit includes the features of its inputs, its outputs, and their related value specifications. The behavior of a module always includes the function(s) to be computed (its semantics), and sometimes the runtime characteristics, such as its space and time complexity. Functional testing derives test data from the features of the specification.

5.1.1.3 Functional Testing Based on the Interface

Functional testing based on the interface of a software module selects test data based on the features of the interfaces. Three types of functional testing are:

- Input Testing

  In external testing, test data is chosen to cover the extremes of the input. Similarly, midrange testing selects data from the interior values. The motivation is inductive – it is hoped that conclusions about the entire input can be drawn from the behavior elicited by some of its representative members. For structured input, combinations of extreme points for each component are chosen. This procedure can generate a large quantity of data.

- Equivalence Partitioning

  Specifications frequently partition the set of all possible inputs into classes that receive equivalent treatment. Such partitioning is called equivalence partitioning. A result of equivalence partitioning is the identification of a finite set of functions and their associated input and output results. Input constraints and error conditions can also result from this partitioning. Once these partitions have been developed, both external and midrange testing are applicable to the resulting input.

- Syntax Checking

  Every program tested must assure its input can handle incorrectly formatted data. Verifying this feature is called syntax checking. One means of accomplishing this is to execute the program using a broad spectrum of test data. By describing the data with documentation language, instances of the input language can be generated using algorithms from automata theory.

5.1.1.4 Functional Testing Based on the Function to be Computed

Equivalence partitioning results in the identification of a finite set of functions and their associated input and output results. Test data can be developed based on the known characteristics of these functions. Consider, for example, a function to be computed that has fixed points, that is, certain of its input values are mapped into themselves by the function. Testing the computation at these fixed points is possible, even in the absence of a complete specification. Knowledge of the function is essential in order to ensure adequate coverage of the output results.

- Special-Value Testing

  Selecting test data on the basis of features of the function to be computed is called special-value testing. This procedure is particularly applicable to mathematical computations. Properties of the function to be computed can aid in selecting points that will indicate the accuracy of the computed solution.

- Output Result Coverage

  For each function determined by equivalence partitioning there is an associated output result. Selecting points that will cause the extremes of each of the output results to be achieved performs output result coverage. This ensures that modules have been checked for maximum and minimum output conditions and that all categories of error messages have, if possible, been produced. In general, constructing such test data

requires knowledge of the function to be computed and, hence, expertise in the application area.

5.1.1.5 Functional Testing Dependent on the Specification Technique

The specification technique employed can aid in testing. An executable specification can be used as an oracle and, in some cases, as a test generator. Structural properties of a specification can guide the testing process. If the specification falls within certain limited classes, properties of those classes can guide the selection of test data.

- Algebraic

  In algebraic specifications, properties of a data abstraction are expressed by means of axioms or rewrite rules. In one system, testing checks the consistency of an algebraic specification with an implementation. Each axiom is compiled into a procedure, which is then associated with a set of test points. A driver program supplies each of these points to the procedure of its respected axiom. The procedure, in turn, indicates whether the axiom is satisfied. Structural coverage of both the implementation and the specification is computed.

- Axiomatic

  Despite the potential for widespread use of predicate calculus as a specification language, little has been published about deriving test data from such specifications. A relationship between predicate calculus specifications and path testing has been explored.

- State Machines

  Many programs can be specified as state machines, thus providing an additional means of selecting test data. Since the equivalence problem of two finite automata is decidable, testing can be used to decide whether a program that simulates a finite automation with a bounded number of nodes is equivalent to the one specified. This result can be used to test those features of programs that can be specified by finite automata, for example, the control flow of a transaction-processing system.

- Decision Tables

  Decision tables are a concise method of representing an equivalence partitioning. The rows of a decision table specify all the conditions that the input may satisfy. The columns specify different sets of actions that may occur. Entries in the table indicate whether the actions should be performed if a condition is satisfied. Typical entries are, “Yes,” “No,” or “Don’t care.” Each row of the table suggests significant test data. Cause-effect graphs provide a systematic means of translating English specifications into decision tables, from which test data can be generated.

5.1.2 Structural Test Cases

In structural program testing and analysis, test data is developed or evaluated from the source code. The goal is to ensure that various characteristics of the program are adequately covered.

5.1.2.1 Structural Analysis

In structural analysis, programs are analyzed without being executed. The techniques resemble those used in compiler construction. The goal here is to identify fault-prone code, to discover anomalous circumstances, and to generate test data to cover specific characteristics of the program's structure.

- Complexity Measures

  As resources available for testing are always limited, it is necessary to allocate these resources efficiently. It is intuitively appealing to suggest that the more complex the code, the more thoroughly it should be tested. Evidence from large projects seems to indicate that a small percentage of the code typically contains the largest number of errors. Various complexity measures have been proposed, investigated, and analyzed.

- Data Flow Analysis

  A program can be represented as a flow graph annotated with information about variable definitions, references, and indefiniteness. From this representation, information about data flow can be deduced for use in code optimization, anomaly detection, and test data generation. Data flow anomalies are flow conditions that deserve further investigation as they may indicate problems. Examples include: defining a variable twice with no intervening reference, referencing a variable that is undefined, and un-defining a variable that has not been referenced since its last definition. Data flow analysis can also be used in test data generation, exploiting the relationship between points where variables are defined and points where they are used.

- Symbolic Execution

  A symbolic execution system accepts three inputs: a program to be interpreted; symbolic input for the program; and the path to follow. It produces two outputs: the symbolic output that describes the computation of the selected path, and the path condition for that path. The specification of the path can be either interactive or pre-selected. The symbolic output can be used to prove the program correct with respect to its specification and the path condition can be used for generating test data to exercise the desired path. Structured data types cause difficulties, however, since it is sometimes impossible to deduce what component is being modified in the presence of symbolic values.

5.1.2.2 Structural Testing

Structural testing is a dynamic technique in which test data selection and evaluation are driven by the goal of covering various characteristics of the code during testing. Assessing such

coverage involves the instrumentation of the code to keep track of which characteristics of the program text are actually exercised during testing. The inexpensive cost of such instrumentation has been a prime motivation for adopting this technique. More importantly, structural testing addresses the fact that only the program text reveals the detailed decisions of the programmer. For example, for the sake of efficiency, a programmer might choose to implement a special case that appears nowhere in the specification. The corresponding code will be tested only by chance using functional testing, whereas use of a structural coverage measure such as statement coverage should indicate the need for test data for this case. Structural coverage measures for a rough hierarchy, with a higher level being more costly to perform and analyze, but being more beneficial, as described below.

- Statement Testing

  Statement testing requires that every statement in the program be executed. While it is obvious that achieving 100 percent statement coverage does not ensure a correct program, it is equally obvious that anything less means that there is code in the program that has never been executed!

- Branch Testing

  Achieving 100 percent statement coverage does not ensure that each branch in the program flow graph has been executed. For example, executing an “if…then” statement, (no “else”) when the tested condition is true, tests only one of two branches in the flow graph. Branch testing seeks to ensure that every branch has been executed. Branch coverage can be checked by probes inserted at points in the program that represent arcs from branch points in the flow graph. This instrumentation suffices for statement coverage as well.

- Conditional Testing

  In conditional testing, each clause in every condition is forced to take on each of its possible values in combination with those of other clauses. Conditional testing thus subsumes branch testing; and therefore, inherits the same problems as branch testing. Instrumentation for conditional testing can be accomplished by breaking compound conditional statements into simple conditions and nesting the resulting “if” statements.

- Expression Testing

  Expression testing requires that every expression assume a variety of values during a test in such a way that no expression can be replaced by a simpler expression and still pass the test. If one assumes that every statement contains an expression and that conditional expressions form a proper subset of all the program expressions, then this form of testing properly subsumes all the previously mentioned techniques. Expression testing does require significant run-time support for the instrumentation.

- Path Testing

  In path testing, data is selected to ensure that all paths of the program have been executed. In practice, of course, such coverage is impossible to achieve for a variety of reasons. First, any program with an indefinite loop contains an infinite number of paths, one for each iteration of the loop. Thus, no finite set of data will execute all paths. The second difficulty is the infeasible path problem: it is undecided whether an

arbitrary path in an arbitrary program is executable. Attempting to generate data for such infeasible paths is futile, but it cannot be avoided. Third, it is undecided whether an arbitrary program will halt for an arbitrary input. It is therefore impossible to decide whether a path is finite for a given input.

In response to these difficulties, several simplifying approaches have been proposed. Infinitely many paths can be partitioned into a finite set of equivalence classes based on characteristics of the loops. Boundary and interior testing require executing loops zero times, one time, and if possible, the maximum number of times. Linear sequence code and jump criteria specify a hierarchy of successively more complex path coverage.

Path coverage does not imply condition coverage or expression coverage since an expression may appear on multiple paths, but some sub-expressions may never assume more than one value. For example, in “if a / b then S1 else S2;” b may be false and yet each path may still be executed.

5.1.3 Erroneous Test Cases

Testing is necessitated by the potential presence of errors in the programming process. Techniques that focus on assessing the presence or absence of errors in the programming process are called error-oriented. There are three broad categories of such techniques: statistical assessment, error-based testing, and fault-based testing. These are stated in order of increasing specificity of what is wrong with the program without reference to the number of remaining faults.

Error-based testing attempts to show the absence of certain errors in the programming process. Fault-based testing attempts to show the absence of certain faults in the code. Since errors in the programming process are reflected as faults in the code, both techniques demonstrate the absence of faults. They differ, however, in their starting point: Error–based testing begins with the programming process, identifies potential errors in that process, and then asks how those errors are reflected as faults. It then seeks to demonstrate the absence of those reflected faults. Fault-based testing begins with finding potential faults in the code regardless of what error in the programming process caused them.

5.1.3.1 Statistical Methods

Statistical testing employs statistical techniques to determine the operational reliability of the program. Its primary concern is how faults in the program affect its failure rate in its actual operating environment. A program is subjected to test data that statistically models the operating environment, and failure data is collected. From the data, a reliability estimate of the program’s failure rate is computed. This method can be used in an incremental development environment. A statistical method for testing paths that compute algebraic functions has also been developed. There has been a prevailing sentiment that statistical testing is a futile activity, since it is not directed toward finding errors. However, studies suggest it is a viable

alternative to structural testing. Combining statistical testing with an oracle appears to represent an effective trade-off of computer resources for human time.

5.1.3.2 Error-Based Testing

Error-based testing seeks to demonstrate that certain errors have not been committed in the programming process. Error-based testing can be driven by histories of programmer errors, measures of software complexity, knowledge of error-prone syntactic constructs, or even error guessing. Some of the more methodical techniques are described below.

- Fault Estimation

  Fault seeding is a statistical method used to assess the number and characteristics of the faults remaining in a program. Harlan Mills originally proposed this technique, and called it error seeding. First, faults are seeded into a program. Then the program is tested and the number of faults discovered is used to estimate the number of faults yet undiscovered. A difficulty with this technique is that the faults seeded must be representative of the yet-undiscovered faults in the program. Techniques for predicting the quantity of remaining faults can also be based on a reliability model.

- Input Testing

  The input of a program can be partitioned according to which inputs cause each path to be executed. These partitions are called paths. Faults that cause an input to be associated with the wrong path are called input faults. Other faults are called computation faults. The goal of input testing is to discover input faults by ensuring that test data limits the range of undetected faults.

- Perturbation Testing

  Perturbation testing attempts to decide what constitutes a sufficient set of paths to test. Faults are modeled as a vector space, and characterization theorems describe when sufficient paths have been tested to discover both computation and input errors. Additional paths need not be tested if they cannot reduce the dimensionality of the error space.

- Fault-Based Testing

  Fault-based testing aims at demonstrating that certain prescribed faults are not in the code. It functions well in the role of test data evaluation. Test data that does not succeed in discovering the prescribed faults is not considered adequate. Fault-based testing methods differ in both extent and breadth. One with local extent demonstrates that a fault has a local effect on computation; it is possible that this local effect will not produce a program failure. A method with global extent demonstrates that a fault will cause a program failure. Breadth is determined by whether the technique handles a finite or an infinite class of faults. Extent and breadth are orthogonal, as evidenced by the techniques described below.

- Local Extent, Finite Breadth

  Input-output pairs of data are encoded as a comment in a procedure, as a partial specification of the function to be computed by that procedure. The procedure is then

executed for each of the input values and checked for the output values. The test is considered adequate only if each computational or logical expression in the procedure is determined by the test; i.e., no expression can be replaced by a simpler expression and still pass the test. Simpler is defined in a way that allows only a finite number of substitutions. Thus, as the procedure is executed, each possible substitution is evaluated on the data state presented to the expression. Those that do not evaluate the same as the original expression are rejected. The system allows methods of specifying the extent to be analyzed.

- Global Extent, Finite Breadth

  In mutation testing, test data adequacy is judged by demonstrating that interjected faults are caught. A program with interjected faults is called a mutant, and is produced by applying a mutation operator. Such an operator changes a single expression in the program to another expression, selected from a finite class of expressions. For example, a constant might be incremented by one, decremented by one, or replaced by zero, yielding one of three mutants. Applying the mutation operators at each point in a program where they are applicable forms a finite, albeit large, set of mutants. The test data is judged adequate only if each mutant in this set is either functionally equivalent to the original program or computes different output than the original program. Inadequacy of the test data implies that certain faults can be introduced into the code and go undetected by the test data.

  Mutation testing is based on two hypotheses. The competent programmer hypothesis says that a competent programmer will write code that is close to being correct. The correct program, if not the current one, can be produced by some straightforward syntactic changes to the code. The coupling effect hypothesis says that test data that reveals simple faults will uncover complex faults as well. Thus, only single mutants need be eliminated, and combinatory effects of multiple mutants need not be considered. Studies formally characterize the competent programmer hypothesis as a function of the probability of the test set's being reliable, and show that under this characterization, the hypothesis does not hold. Empirical justification of the coupling effect has been attempted, but theoretical analysis has shown that it does not hold, even for simple programs.

- Local Extent, Infinite Breadth

  Rules for recognizing error-sensitive data are described for each primitive language construct. Satisfaction of a rule for a given construct during testing means that all alternate forms of that construct have been distinguished. This has an obvious advantage over mutation testing – elimination of all mutants without generating a single one! Some rules even allow for infinitely many mutants. Of course, since this method is of local extent, some of the mutants eliminated may indeed be the correct programs.

- Global Extent, Infinite Breadth

  We can define a fault-based method based on symbolic execution that permits elimination of infinitely many faults through evidence of global failures. Symbolic faults are inserted into the code, which is then executed on real or symbolic data.

Program output is then an expression in terms of the symbolic faults. It thus reflects how a fault at a given location will impact the program's output. This expression can be used to determine actual faults that could not have been substituted for the symbolic fault and remain undetected by the test.

5.1.4 Stress Test Cases

Stress or volume testing needs a tool that supplements test data. The objective is to verify that the system can perform properly when stressed, or when internal program or system limitations have been exceeded. This may require that large volumes of test cases be entered during testing. Stressing can be done with large volumes of data on a test file; however, a tool will be needed to test using a script.

The types of internal limitations that can be evaluated with volume testing include:

- Internal accumulation of information, such as tables.
- Number of line items in an event, such as the number of items that can be included within an order.
- Size of accumulation fields.
- Data-related limitations, such as leap year, decade change, switching calendar years, etc.
- Field size limitations, such as number of characters allocated for people's names.
- Number of accounting entities, such as number of business locations, state/country in which business is performed, etc.

The concept of stress testing is as old as the processing of data in information systems. What is necessary to make the concept work is a systematic method of identifying limitations. The recommended steps for determining program and system limitations follow.

1. Identify input data used by the program.

   A preferred method to identify limitations is to evaluate the data. Each data field is reviewed to determine if it poses a system limitation. This is an easier method than attempting to evaluate the programs. The method is also helpful in differentiating between system and program limitations. It also has the advantage that data may only need to be evaluated once, rather than evaluating numerous individual programs.

   All of the data entering an application system should be identified. Those data elements not used by the applications, but merely accessible to it, should be deleted, resulting in a list of input data used by the application system.

2. Identify data created by the program.

   Data generated by application systems should be identified. These would be data elements that are not inputted into the system but are included in internal or output data records. Knowing the input data and the output data, it is a relatively simple process to identify newly created data elements.

3. Challenge each data element for potential limitations.

A key step in determining program/system limitations is in the challenge process. The individual using stress test tools should ask the following questions about each data element:

- Can the data value in a field entering the system exceed the size of this data element? (If so, a limitation is identified.)
- Is the value in a data field accumulated? (If so, a limitation is identified.)
- Is data temporarily stored in the computer? (If so, a limitation is identified.)
- Is the information in a data element(s) stored in the program until a following transaction is entered? (If so, a limitation is identified.)
- If a data element represents an accounting entity, for example, the number of sales financial accounts, etc., does the number used to identify the accounting entity in itself provide a future limitation, such as using a one-character field to identify sales districts? (If so, a limitation is identified.)

4. Document limitations.

   All of the limitations identified in Step 3 should be documented. This forms the basis for stress testing. Each of these limitations must now be evaluated to determine the extent of testing on those limitations.

5. Perform stress testing.

   The testing to be performed follows the same nine steps outlined in the test file process. The limitations documented in Step 4 become the test conditions that need to be identified in Step 2.

5.1.5 Test Scripts

Test scripts are an on-line entry of test cases in which the sequence of entering test cases and the structure of the on-line entry system must be validated, in addition to the expected results from a single test case.

The following tasks are needed to develop, use, and maintain test scripts:

- Determine testing levels
- Develop the scripts
- Execute the scripts
- Analyze the results
- Maintain the scripts

5.1.5.1 Determine Testing Levels

- *Unit Scripting* – Develop a script to test a specific unit or module.
- *Pseudo-concurrency Scripting* – Develop scripts to test when there are two or more users accessing the same file at the same time.
- *Integration Scripting* – Determine that various modules can be properly linked.

- *Regression Scripting* – Determine that the unchanged portions of systems remain unchanged when the system is changed. (Note: This is usually performed with the information captured on a capture/playback software system that enables the capture of transactions as they are entered via terminal, and then repeats them as the scripts are reused. There are many of these on the market, although they are aimed primarily at the IBM mainframe.)
- *Stress and Performance Scripting* – Determine whether the system will perform correctly when it is stressed to its capacity. This validates the performance of the software when stressed by large numbers of transactions.

The testers need to determine which, or all, of these five levels of scripting to include in the script.

5.1.5.2 Develop the Scripts

This task, too, is normally done using the capture and playback tool. The script is a complete series of related terminal actions. The development of a script involves a number of considerations, as follows:

- Script components
- Programs to be tested
- On-line operating environment
- Manual entry of script transactions
- Secured initialization
- Password entry
- Automated entry of script transactions
- Inquiry during processing
- Stress and performance test
- File states and contents
- Operating environment
- Complete scripts
- Start; usually begins with a clear screen
- Scripts; end with a clear screen
- Sign-on
- Menu navigation
- Exit
- Clear screen
- Regrouping
- Sources of scripting transactions
- Operations initialization of files
- Inquiry versus update
- Input
- Files involved
- Output
- Date setup
- File restores
- Update
- Edits of transactions
- External considerations
- Program libraries
- Screen initialization
- Security considerations
- Start and stop considerations
- Start; begins with a transaction code
- Script contents
- Setup
- Function
- Sign-off
- Changing passwords
- Single-user identifications
- Entry of scripts
- Timing dependencies
- Unit versus regression test

- Organization of scripts
- Single functions (transactions)
- Separate inquiry from update
- Pseudo-concurrent test
- Multiple functions (transactions)
- Special considerations
- Single versus multiple terminals
- Unit test organization
- Single terminal
- Self-maintaining
- Integration test (string testing)
- Regression test
- Multiple terminals
- Date and time dependencies

- Three steps: setup (external), test (script), and reset (external)
- Application program interface (API) communications
- Navigation of transactions through the system
- User identification and security rules
- Iterative/vary arrival rate; three steps: setup (external), test (script), and collect performance data

The following is needed for script development:

- Test Item – a unique item identified of the test condition.
- Entered by – Who will enter the script.
- Sequence – The sequence in which the actions are to be entered.
- Action – The action or scripted item to be entered.
- Expected Result – The result expected from entering the action.
- Operator Instructions – What the operator is to do if the proper result is received, or if an improper result is returned.

Table 5-1summarizes the script development strategies. The table shows for the five levels of testing using scripts described above, which level is best suited for single transaction tests, and which is best suited for testing multiple transactions. The table also shows for each level whether testing occurs from a single terminal or from multiple terminals.

Table 5-1: Script Development Strategies

Test Level	Single Transaction	Multiple Transactions	Single Terminal	Multiple Terminals
Unit	X		X	
Concurrent	X			X
Integration		X	X	
Regression		X		X
Stress		X		X

5.1.5.3 Execute the Script

The script can be executed manually or by using the capture and playback tools. Use caution when you use scripting extensively unless a software tool drives the script. Some of the considerations to incorporate into script execution are:

- Environmental setup
- Program libraries
- File states and contents
- Date and time
- Security
- Multiple terminal arrival modes
- Think time
- Serial (cross-terminal) dependencies
- Pseudo-concurrent
- Processing options
- Stall detection
- Synchronization
- Rate
- Arrival rate

5.1.5.4 Analyze the Results

After executing the test script, the results must be analyzed. However, much of this should have been done during the execution of the script, using the operator instructions provided. Note: if a capture and playback software tool is used, analysis will be more extensive after execution. The result analysis should include the following:

- System components
- Outputs (screens)
- File content at conclusion of testing
- Status of logs

- Performance data (stress results)
- On-screen outputs
- Individual screen outputs
- Multiple screen outputs
- Order of outputs processing
- Compliance of screens to specifications
- Ability to process actions
- Ability to browse through data

The main checking will be the actual results against the expected results. The preceding list highlights some of the specific considerations included.

5.1.5.5 Maintain Scripts

Once developed, scripts need to be maintained so that they can be used throughout development and maintenance. The areas to incorporate into the script maintenance procedure are:

- Programs
- Files
- Screens
- Insert (transactions)
- Delete
- Arrange
- Field
- Changed (length, content)
- New
- Moved
- Expand test cases

Use Table 5-2, “Test Condition Script Checklist” to determine the completeness of the scripts. If the script does not address items in the checklist, you should consider extending the script. A “No” response indicates that you have not included a test condition of that type, and you may want to indicate the reason for not including this test condition. The following questions are prefaced with “Does the script include…”:

Table 5-2: Test Condition Script Checklist

Does the Script Include? …	Yes	No	If No, Why Not?
Unit testing			
Pseudo-concurrency testing			
Integration testing			
Regression testing			
Stress testing			
Manual entries of transactions for:			
Date setup			
Secured initialization			
File restores			
Password entry			
Updates			
Automated entries of scripts include the appropriate navigation through the system			
References to the program libraries			
Various file states			
Initialization of screens			
The operating environment			
Security considerations			
Sign-on and setup procedures			
Sign-off and clear screen procedures			
All items on the menu			
Changing user identification			
Changing passwords			
Use of the prompting routines			
Multiple terminal processing			
Time and date dependencies			
Single-function transactions			
Multiple-function transactions			
Inquiry			
Update			
Deletion			

Does the Script Include? …	Yes	No	If No, Why Not?
Multiple-terminal testing			
Single-terminal testing			
Single-transaction testing			
Multiple-transactions testing			

The output from this workbench is a description of the results of the scripting test, which might include:

- Performance and problems in manual entry
- Performance and problems in the data entry process
- Performance and problems associated with the hardware, menu navigation, sign-on and sign-off procedures
- Problems with quality characteristics such as ease of use and security
- Capability of the on-line system to perform in accordance with specifications.

Several characteristics of scripting are different from batch test data development. These differences are:

- Data entry procedures required

  The test procedures take on greater significance in scripting. The person using the script needs to know in detail how to enter the transaction via the terminal. This may be more complex than simply creating a test condition.

- Use of software packages

  Scripting is a very difficult and complex task to do manually, particularly when the script has to be repeated multiple times. Therefore, most testers use a capture/playback type of software package.

- Sequencing of events

  Scripts require the sequencing of transactions. In batch systems, sequencing is frequently handled by sorting during systems execution; however, with scripts, the sequence must be predefined.

- Stop procedures

  Batch testing continues until the batch is complete or processing abnormally terminates. Scripting may be able to continue, but the results would be meaningless; therefore, the script has to indicate when to stop, or if specific conditions occur, where to go in the script to resume testing.

5.1.6 Use Cases

Incomplete, incorrect, and missing test cases can cause incomplete and erroneous test results. Flawed test results causes rework, at a minimum, and at worst, a flawed system to be

developed. There is a need to ensure that all required test cases are identified so that all system functionality requirements are tested.

A use case is a description of how a user (or another system) uses the system being designed to perform a given task. A system is described by the sum of its use cases. Each instance or scenario of a use case will correspond to one test case. Incorporating the use case technique into the development life cycle will address the effects of incomplete, incorrect, and missing test cases. Use cases are an easy-to-use approach that is applicable to both conventional and object-oriented system development.

Use cases provide a powerful means of communication between customer, developers, testers, and other project personnel. Test cases can be developed with system users and designers as the use cases are being developed. Having the test case developed at the earliest possible stage of the project provides a baseline for the early planning of acceptance testing. Another advantage to having test cases early on is that, if a packaged software solution is indicated, then the customer can use them to evaluate purchased software earlier in the development cycle. Using the use case approach will ensure not only meeting requirements, but also expectations.

5.1.6.1 Build a System Boundary Diagram

A system boundary diagram depicts the interfaces between the software being tested and the individuals, systems, and other interfaces. These interfaces or external agents in this work practice will be referred to as “actors.” The purpose of the system boundary diagram is to establish the scope of the system and to identify the actors (i.e., the interfaces) that need to be developed.

An example of a system boundary diagram for an automated teller machine (ATM) for an organization called “Best Bank” is illustrated in Figure 5-1.

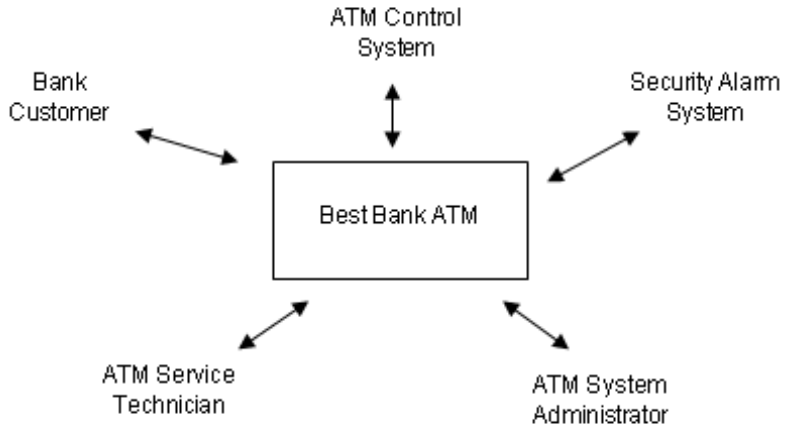


Figure 5-1 System Boundary Diagram for an ATM Example

For that software, each system boundary needs to be defined. System boundaries can include:

- Individuals/groups that manually interface with the software
- Other systems that interface with the software
- Libraries
- Objects within object-oriented systems

Each system boundary should be described. For each boundary an actor must be identified.

Two aspects of actor definition are required. The first is the actor description, and the second is the name of an individual or group who can play the role of the actor (i.e., represent that boundary interface). For example, in Figure 5-1 the security alarm system is identified as an interface. The actor is the security alarm company. The name of a person in the security alarm company or the name of someone who can represent the security alarm company must be identified. Note that in some instances the actor and the individual may be the same, such as the ATM system administrator listed in Figure 5-1.

5.1.6.2 Define Use Cases

An individual use case consists of:

- Preconditions that set the stage for the series of events that should occur for the use case
- Results that state the expected outcomes of the above process
- Sequential narrative of the execution of the use case

Use cases are used to:

- Manage (and trace) requirements
- Identify classes and objects (OO)
- Design and code (Non-OO)
- Develop application documentation
- Develop training
- Develop test cases

The use case is defined by the actor. The actor represents the system boundary interface and prepares all of the use cases for that system boundary interface. Note that this can be done by a single individual or a team of individuals.

The information about each use case that needs to be determined for defining the case follows:

- Use Case Name or ID

  A short phrase in business terms or identifier that identifies and describes the use case.

- Actor

  Anything that needs to exchange information with the system. Often it is a role played by the user of the system or it could be another system.

- Objective

  A description of what a use case accomplishes given a defined set of conditions.

- Preconditions

  The entrance criteria or state that the system must be in for the use case to execute.

- Results

  The expected completion criteria of the use case.

- Detailed Description
  - The sequence of steps (performed by the actor) necessary to execute the use case.
  - The model (system) response of each step.
  - This is the basic course of events that support the precondition and results.
  - The description is from a user's ("black-box") point of view and does not include details about the events of the internal system.
- Exceptions
  - Errors or deviations that may occur that cause the actor to deviate from the basic course.
- Alternative Courses
  - A deviation from the step-by-step event so the Detailed Description that generally inserts extra steps or omits steps.
  - These are valid events, but are not the basic course of events.

5.1.6.3 Develop Test Cases

A test case is a set of test inputs, execution conditions, and expected results developed for a particular test objective. There should be a one-to-one relationship between use case definitions and test cases. There needs to be at least two test cases for each use case: one for successful execution of the use case and one for an unsuccessful execution of a test case. However, there may be numerous test cases for each use case.

Additional test cases are derived from the exceptions and alternative course of the use case. Note that additional detail may need to be added to support the actual testing of all the possible scenarios of the use case.

The use case description is the input to the test case worksheet. The actor who prepared the use case description also prepares the test case worksheet. There will be at least two test conditions for each use case description and normally many more. The actor tries to determine all of the possible scenarios that occur for each use case. To build a use case, the following needs to be determined:

5.1.6.4 Test Objective

The specific objective of the test case. The test objective is related to the use case definition that details the description action.

- Test Condition

  One of the possible scenarios as a result of the action being tested from the use case description worksheet.

- Operator Action

  The detailed steps that the operator performing the test condition performs to execute the test condition.

- Input Specifications

  The input that is necessary in order for the test case to be executed.

- Output Specifications

  The results expected from performing the operator actions on the input specified

- Pass or Fail

  The results of executing the test.

- Comments

  Guidance from the actor to the individual who will actually perform the test.

At the conclusion of use case testing, a decision must be made on each acceptance criterion as to whether it has been achieved.

5.1.7 Building Test Cases

Testers and users have been using test cases since the inception of computer programming. The concept of test data is a simple one – creating representative processing conditions using test cases. The complex part of creating test data is determining which processing events to make test cases. Experience shows that it is uneconomical to test all conditions in an application system. Experience further shows that most testing exercises less than one-half of the computer instructions. Therefore, optimizing testing through selecting the most important processing events is the key aspect of building test cases.

Several of the test tools are structured methods for designing test cases. For example, correctness proof, data flow analysis, and control flow analysis are all designed to develop extensive sets of test cases, as is exhaustive testing. Unfortunately, all of these tools, while extremely effective, require large amounts of time and effort to implement. Few organizations allocate sufficient budgets for this type of testing. In addition, many IT personnel are not trained in the use of these test tools.

Test cases can be built as follows:

- Built manually

- Created by a test case generator
- Extracted from a production file

5.1.8 Process for Building Test Cases

The recommended process for the creation and use of test cases is a nine-step process as follows:

1. Identify test resources.

   Testing using test cases can be as extensive or limited a process as desired. Unfortunately, many programmers approach the creation of test data from a "we'll do the best job possible" perspective and then begin developing test transactions. When time expires, testing is complete. The recommended approach suggests that the amount of resources allocated for the test data test tool is determined and then a process developed that optimizes that time.

2. Identify conditions to be tested.

   A testing matrix is recommended as the basis for identifying conditions to test. As these matrices cascade through the developmental process, they identify all possible test conditions. If the matrix concept is not used, then the possible test conditions should be identified during the use of this test tool. These should be general test conditions, such as in a payroll application to test the FICA deductions.

3. Rank test conditions.

   If resources are limited, the maximum use of those resources will be obtained by testing the most important test conditions. The objective of ranking is to identify high-priority test conditions that should be tested first.

   Ranking does not mean that low-ranked test conditions will not be tested. Ranking can be used for two purposes: first, to determine which conditions should be tested first; and second, and equally as important, to determine the amount of resources allocated to each of the test conditions. For example, if testing the FICA deduction was a relatively low-ranked condition, only one test transaction might be created to test that condition, while for the higher ranked test conditions several test transactions may be created.

4. Select conditions for testing.

   Based on the ranking, the conditions to be tested should be selected. At this point, the conditions should be very specific. For example, "testing FICA" is a reasonable condition to identify and rank, but for creating specific test conditions it is too general. Three test situations might be identified – such as employees whose year-to-date earnings exceed the maximum FICA deduction; an employee whose current-period earnings will exceed the difference between the year-to-date earnings and the maximum deduction; and an employee whose year-to-date earnings are more than one pay period amount below the maximum FICA deductions. Each test situation should be documented in a testing matrix. This is a detailed version of the testing matrix that was started during the requirements phase.

5. Determine correct results of processing.

   The correct processing results for each test situation should be determined. A unique number should identify each test situation, and then a log made of the correct results for each test condition. If a system is available to automatically check each test situation, special forms may be needed as this information may need to be converted to machine-readable media.

   The correct time to determine the correct processing results is before the test transactions have been created. This step helps determine the reasonableness and usefulness of test transactions. The process can also show if there are ways to extend the effectiveness of test transactions, and whether the same condition has been tested by another transaction.

6. Create test cases.

   Each test situation needs to be converted into a format suitable for testing. In some instances, this requires the creation of a test case and master information to be stored by the program for the purpose of processing the test case. The method of creating the machine-readable transaction will vary based on the application and the test rules available in the information systems department.

   The most common methods of creating test cases include:

   - Key entry
   - Test data generator
   - Preparation of an input form which will be given to user personnel to enter

7. Document test conditions.

   Both the test situations and the results of testing should be documented.

8. Conduct test.

   The executable system should be run, using the test conditions. Depending on the extent of the test, it can be run under a test condition or in a simulated production environment.

9. Verify and correct.

   The results of testing should be verified and any necessary corrections to the programs performed. Problems detected as a result of testing can be attributable not only to system defects, but to test data defects. The individual conducting the test should be aware of both situations.

5.1.9 Example of Creating Test Cases for a Payroll Application

This example shows a test case development approach of an automated payroll system. First, all available documentation was reviewed for the manual and automated parts of each system. To understand the manual operations, the test team interviewed payroll supervisors and clerks, reviewed laws and regulations relating to pay and leave, and familiarized themselves with standard payroll operating procedures. For the automated part of each system they interviewed system designers and programmers and reviewed system and program documentation and operating procedures.

After acquiring a working knowledge of each system, they decided to test computer programs used to update payroll master records and those used to calculate biweekly pay and leave entitlements. Although they were concerned primarily with these particular programs, they decided that other programs used in the normal biweekly payroll processing cycle (such as programs for producing pay and leave history reports, leave records, and savings bond reports) should also be tested to see how they would handle test data.

They then designed a test file of simulated pay and leave transactions to test the effectiveness of internal controls, compliance with applicable laws and regulations, and the adequacy of standard payroll operating procedures. The test file included transactions made up of both valid and invalid data. These transactions were based on specified procedures and regulations and were designed to check the effectiveness of internal controls in each installation's payroll processing. They used one transaction for each master record chosen.

The best method of obtaining suitable payroll master records for the test, they decided, would be to use copies of actual master records, supplemented with simulated records tailored for test conditions not found in the copied records.

Accordingly, they obtained a duplicate of the payroll master file and had a section of it printed in readable copy. From this printout, they selected a specific master record to go with each test transaction. When none of the copied records appearing on the printout fit the specifics of a particular transaction, they made up a simulated master record by preparing source documents and processing them with the program used by each installation to add records for new employees to its master file. They then added the simulated records to the copied records to create the test master file. The test team then recorded their comparisons of the predetermined results with the actual results.

Examples of test cases for the payroll example are illustrated in Figure 5-2.

			Expected Results						
Control Number	Nature of Test Case	Purpose or Explanation of Test Case	Reject	Print error message	Reject in Certain Circumstances	Cut Back to Allowable Maximum	Process Without Cutback	Automatically Compute Correct Amount	Automatically Adjust Leave Records
1	Leave a mandatory field blank on a new employee's master record.	To determine whether the system will accept a master record with essential data missing. If missing data will cause an incorrect payment, the master record should be rejected with appropriate warning; if missing data is for administrative purposes only, the condition should be flagged by an error message.		X	X				
2	Enter erroneous codes.	To determine whether the system will accept invalid data into employees' master records. The program should print error messages to identify invalid data and reject further processing of such transactions.		X	X				
3	Change a field in an inactive master record.	To determine whether it is possible to change a field in inactive master records and whether adequate controls exist over such changes. Processing of inactive records should be separated from the normal processing of active records to eliminate the possibility of unearned salary payments or the manipulation of records for persons who are not in a pay status.	X	X					
4	Change an employee's grade or annual salary so that the grade/step and annual salary rate are incompatible.	To determine whether the system accepts incompatible data. The program should have salary and grade controls that will reject transactions of this type from further processing.		X	X				
5	Pay an inactive employee.	To determine whether the system will compute pay for an inactive employee (an employee who has been separated but whose record is maintained in the same master file used for active employees).	X	X					
6	Input two time and attendance cards for the same employee.	To determine whether the system will compute pay twice for the same employee.	X	X					
7	Pay an hourly employee for 20 hours overtime.	To verify the accuracy of premium (overtime) pay computation. Overtime pay is 1 and ½ times regular pay.						X	
8	Pay an employee for 20 hours of night-differential pay.	Same as above. Premium = 10 percent.						X	

Figure 5-2 Example of Test Case for a Payroll System

5.2 Test Coverage

Based upon the risk, and criticality associated with the application under test, the project team should establish a coverage goal during test planning. The coverage goal defines the amount of code that must be executed by the tests for the application. In those cases where the application supports critical functions, such as air traffic control or military defense systems, the coverage goal may be 100% at all stages of testing.

The objective of test coverage is simply to assure that the test process has covered the application. Although this sounds simple, effectively measuring coverage may be critical to the success of the implementation. There are many methods that can be used to define and measure test coverage, including:

- Statement Coverage
- Branch Coverage
- Basis Path Coverage
- Integration Sub-tree Coverage
- Modified Decision Coverage
- Global Data Coverage
- User-specified Data Coverage

It is usually necessary to employ some form of automation to measure the portions of the application covered by a set of tests. Tools like McCabe and BattleMap support test coverage analysis in order to both accelerate testing and widen the coverage achieved by the tests. The development team can also design and implement code instrumentation to support this analysis. This automation enables the team to:

- Measure the “coverage” of a set of test cases
- Analyze test case coverage against system requirements
- Develop new test cases to test previously “uncovered” parts of a system

Even with the use of tools to measure coverage, it is usually cost prohibitive to design tests to cover 100% of the application outside of unit testing or black-box testing methods. One way to leverage a dynamic analyzer during system testing is to begin by generating test cases based on functional or black-box test techniques. Examine the coverage reports as test cases are executed. When the functional testing provides a diminishing rate of additional coverage for the effort expended, use the coverage results to conduct additional white-box or structural testing on the remaining parts of the application until the coverage goal is achieved.

5.3 Performing Tests

Test execution is the operation of a test cycle. Each cycle needs to be planned, prepared for, executed and the results recorded. This section addresses these activities involved in performing tests:

- Test platforms
- Test cycle strategy
- Use of tools in testing
- Test execution
- Executing the Unit Test plan
- Executing the Integration Test Plan
- Executing the System Test Plan
- When is Testing Complete?
- Concerns

5.3.1 Platforms

As discussed in Skill Category 2, Building the Test Environment platforms must be established for conducting tests. For example in testing of web-based systems the test environment needs to simulate the type of platforms that would be used in the web environment.

Since the test scripts and test data may need to run on different platforms, the platforms must be taken into consideration in the design of test data and test scripts. Since a large number of platforms may be involved in operation of the software, testers need to decide which platforms are needed for test purposes.

5.3.2 Test Cycle Strategy

Each execution of testing is referred to as a test cycle. Ideally the cycles are planned and included in the test plan. However, as defects are uncovered, and change is incorporated into the software, additional test cycles may be needed.

Software testers should determine the number and purpose of the test cycles to be used during testing. Some of these cycles will focus on the level of testing, for example unit, integration and system testing. Other cycles may address attributes of the software such as data entry, database updating and maintenance, and error processing.

5.3.3 Use of Tools in Testing

Testing, like program development, generates large amounts of information, necessitates numerous computer executions, and requires coordination and communication between workers. Test tools can ease the burden of test design, test execution, general information handling, and communication.

General system utilities and test generation tools are invaluable for test preparation, organization, and modification. A well-organized and structured file system and a good text editor are a minimum support set. A more powerful support set includes data reduction and report generation tools. Library support systems consisting of a database management system and a configuration control system are as useful during testing, as during software development since data organization, access, and control are required for management of test files and reports.

5.3.3.1 Test Documentation

Most guidelines for software documentation during the development phase recommend that test documentation be prepared for all multipurpose or multi-user projects and for other large software development projects. The preparation of a Test Plan and issuing a Test Analysis Report is recommended. The Test Plan should identify test milestones and provide the testing schedule and requirements. In addition, it should include specifications, descriptions, and procedures for all tests, and the test data reduction and evaluation criteria. The Test Analysis Report should summarize and document the test results and findings. The analysis summary should present the software capabilities, deficiencies, and recommendations. As with all types of documentation, the extent, formality, and level of detail of the test documentation are functions of IT standards and may vary depending upon the size, complexity, and risk of the project.

5.3.3.2 Test Drivers

Unless the module being developed is a stand-alone program, considerable auxiliary software must be written in order to exercise and test it. Auxiliary code which sets up an appropriate environment and calls the module is termed a driver, while code which simulates the results of a routine called by the module is a stub. For many modules both stubs and drivers must be written in order to execute a test.

When testing is performed incrementally, an untested function is combined with a tested one and the package is then tested. Such packaging can lessen the number of drivers and/or stubs that must be written. When the lowest level of functions, those which call no other function, are tested first and then combined for further testing with the functions that call them, the need for writing stubs can be eliminated. This approach is called bottom-up testing. Bottom-up testing still requires that test drivers be constructed. Testing which starts with the executive functions and incrementally adds functions which it calls, is termed top-down testing. Top-down testing requires that stubs be created to simulate the actions of called functions that have

not yet been incorporated into the system. The testing order utilized should be coordinated with the development methodology used.

5.3.3.3 Automatic Test Systems and Test Languages

The actual performance of each test requires the execution of code with input data, an examination of the output, and a comparison of the output with the expected results. Since the testing operation is repetitive in nature, with the same code executed numerous times with different input values, an effort has been made to automate the process of test execution. Programs that perform this function of initiation are called test drivers, test harnesses, or test systems.

The simplest test drivers merely re-initiate the program with various input sets and save the output. The more sophisticated test systems accept data inputs, expected outputs, the names of routines to be executed, values to be returned by called routines, and other parameters. These test systems not only initiate the test runs but also compare the actual output with the expected output and issue concise reports of the performance.

A side benefit of a comprehensive test system is that it establishes a standard format for test materials, which is extremely important for regression testing. Currently, automatic test driver systems are expensive to build and consequently are not in widespread use.

5.3.4 Perform Tests

In a life cycle approach to testing, test performance can occur throughout the project life cycle, from testing requirements through conducting user acceptance testing. This discussion will focus on the performance of the dynamic testing that is planned for an application.

The more detailed the test plan, the easier this task becomes for the individuals responsible for performing the test. The test plan should have been updated throughout the project in response to approved changes made to the application specifications. This process ensures that the true expected results have been documented for each planned test.

The roles and responsibilities for each stage of testing should also have been documented in the test plan. For example, the development team might be responsible for unit testing in the development environment, while the test team is responsible for integration and system testing.

The Test Manager is responsible for conducting the Test Readiness Review prior to the start of testing. The purpose of this review is to ensure that all of the entrance criteria for the test phase have been met, and that all test preparations are complete.

The test plan should contain the procedures, environment, and tools necessary to implement an orderly, controlled process for test execution, defect tracking, coordination of rework, and configuration & change control. This is where all of the work involved in planning and set-up pays off.

For each phase of testing, the planned tests are performed and the actual results are compared to the documented expected results. When an individual performs a test script, they should be aware of the conditions under test, the general test objectives, as well as specific objectives listed for the script. All tests performed should be logged on a Test Execution Log, or in a tool such as Mercury's Test Director, by the individual performing the test.

The Test Log is a simple worksheet or spreadsheet that records test activities in order to maintain control over the test. It includes the test ID, test activities, start and stop times, pass or fail results, and comments. Be sure to document actual results. Log the incidents into the defect tracking system once a review determines it is actually a defect.

When the development team communicates the defect resolution back to the test team, and the fix is migrated to the test environment, the problem is ready for retest and execution of any regression testing associated with the fix.

5.3.4.1 Perform Unit Testing

Unit testing is normally performed by the programmer that developed the program. Unit testing can be performed many ways but the result of unit testing should be that the unit is defect free. In other words, the program performs as specified. Integration testing should not occur until the units included in integration testing are defect free.

5.3.4.2 Perform Integration Test

Integration testing should begin once unit testing for the components to be integrated is complete, and should follow the basic testing process outlined in the previous section. The objectives in this stage of testing are to validate the application design, and prove that the application components can be successfully integrated to perform one or more application functions. The team must also prove that the application integrates correctly into its environment.

For client/server applications, this process may involve multiple iterations through the integration test process. The most effective method for validating successful integration is to:

- Test the client components
- Test the server components
- Test the network
- Integrate the client, server, and network

Some common items to focus on during Integration testing include:

- Validation of the links between the client and server(s)
- Security controls
- Performance and load tests on individual application components such as the database, network, and application server
- Sequences of adds, updates, views, and deletes within the application

- Simple transaction completion
- Tests for empty database or file conditions
- Output interface file accuracy
- Back-out situations

Depending on the sequence and design of the integration test "builds," the application may be ready to enter System Test once the pre-defined exit criteria have been met.

5.3.4.3 Perform System Test

System test should begin as soon as a minimal set of components has been integrated and successfully completed integration testing. System test ends when the test team has measured system capabilities and corrected enough of the problems to have confidence that the system will operate successfully in production.

Once test planning is complete, preparation activities and actual test execution begins. Although many activities may be included in this process, the major steps are outlined below:

- Set up system test environment, mirroring the planned production environment as closely as possible.
- Establish the test bed.
- Identify test cases that will be included in the system test.
- Identify test cycles needed to replicate production where batch processing is involved.
- Assign test cases to test cycles; note that in applications where the processing is real-time the test sets may still need to be grouped by planned days if the sequence of test execution is critical.
- Assign test scripts to testers for execution.
- Review test results and determine whether problems identified are actually defects.
- Record defect(s) in a tracking system, making sure the developer responsible for fixing the defect(s) is notified.
- When a defect is fixed and migrated to the test environment, re-test and validate the fix. If the defect is fixed, close the defect log. If the defect is not fixed, return it to the developer for additional work.

The system test focuses on determining whether the requirements have been implemented correctly. This includes verifying that users can respond appropriately to events such as month-end processing, year-end processing, business holidays, promotional events, transaction processing, error conditions, etc.

In client/server testing, the test team must also prove that the application runs successfully on all supported hardware and software environments. This becomes even more complex with Internet applications that must also support various versions of the supported browsers. Standard backup and recovery procedures must also be tested, as well as special security requirements.

5.3.5 When is Testing Complete?

How do you know when testing is complete? Most testers might answer, “When I run out of time!” but there are other factors the Test Manager can use to make this decision. The Test Manager must be able to report, with some degree of confidence, that the application will perform as expected in production and whether the quality goals defined at the start of the project have been met.

The Test Manager may use a set of test metrics, including Mean Time Between Failure or the percentage of coverage achieved by the executed tests, to determine whether the application is ready for production. Other factors, such as the number of open defects and their severity levels, must also be taken into consideration. Finally, the risk associated with moving the application into production, as well as the risk of not moving forward, must be addressed.

5.3.6 General Concerns

There are three general concerns testers have in performing tests:

- Software is not in a testable mode for this test level.

  The previous testing levels will not have been completed adequately to remove most of the defects and the necessary functions will not have been installed, or correctly installed in the software (i.e., performing integration testing when some units contain defects). Thus, testing will become bogged down in identifying problems that should have been identified earlier.

- There is inadequate time and resources.

  Because of delays in development or failure to adequately budget sufficient time and resources for testing, the testers will not have the time or resources necessary to effectively test the software. In many IT organizations, management relies on testing to assure that the software is ready for production prior to being placed in production. When adequate time or resources are unavailable, management may still rely on the testers when they are unable to perform their test as expected.

- Significant problems will not be uncovered during testing.

  Unless testing is adequately planned and executed according to that plan, the problems that can cause serious operational difficulties may not be uncovered. This can happen because testers at this step spend too much time uncovering defects rather than evaluating the operational performance of the application software.

5.4 Recording Test Results

A test problem is a condition that exists within the software system that needs to be addressed. Carefully and completely documenting a test problem is the first step in correcting the

problem. While the word “problem” is used in this practice, some software testers refer to these problems as “defects.”

The following four attributes should be developed for all test problems:

- Statement of condition – Tells what is.
- Criteria – Tells what should be.

  Please note that the two attributes above are the basis for a finding. If a comparison between the two gives little or no practical consequence, no finding exists.

- Effect – Tells why the difference between what is and what should be is significant.
- Cause – Tells the reasons for the deviation. Identification of the cause is necessary as a basis for corrective action.

A well-developed problem statement will include each of these attributes. When one or more of these attributes is missing, questions usually arise, such as:

- Criteria – Why is the current state inadequate?
- Effect – How significant is it?
- Cause – What could have caused the problem?

Documenting a statement of a user problem involves three sub-tasks, which are explained in the following paragraphs.

5.4.1 Problem Deviation

Problem statements begin to emerge by a process of comparison. Essentially the user compares “what is” with “what should be.” When a deviation is identified between what is found to actually exist and what the user thinks is correct or proper, the first essential step toward development of a problem statement has occurred. It is difficult to visualize any type of problem that is not in some way characterized by this deviation. The “what is” can be called the statement of condition. The “what should be” shall be called the criteria. These concepts are the first two, and most basic, attributes of a problem statement.

The documenting of deviation is describing the conditions, as they currently exist, and the criteria, which represents what the user desires. The actual deviation will be the difference or gap between “what is” and “what is desired.”

The statement of condition is uncovering and documenting the facts, as they exist. What is a fact? If somebody tells you something happened, is that “something” a fact? On the other hand, is it only a fact if someone told you it’s a fact? The description of the statement of condition will of course depend largely on the nature and extent of the evidence or support that is examined and noted. For those facts making up the statement of condition, the IT professional will obviously take pains to be sure that the information is accurate, well supported, and worded as clearly and precisely as possible.

The statement of condition should document as many of the following attributes as appropriate for the problem:

- Activities involved – The specific business or administrated activities that are being performed.
- Procedures used to perform work – The specific step-by-step activities that are utilized in producing output from the identified activities.
- Outputs/Deliverables – The products that are produced from the activity.
- Inputs – The triggers, events, or documents that cause this activity to be executed.
- User/Customers served – The organization, individuals, or class users/customers serviced by this activity.
- Deficiencies noted – The status of the results of executing this activity and any appropriate interpretation of those facts.

The criterion is the user's statement of what is desired. It can be stated in either negative or positive terms. For example, it could indicate the need to reduce the complaints or delays as well as desired processing turnaround time.

There are often situations where what "should be," can relate primarily to common sense or general reasonableness, and the statement of condition virtually speaks for itself. These situations must be carefully distinguished from personal whims or subjective, fanciful notions. There is no room for such subjectivity in defining what is desired.

As much as possible the criteria should be directly related to the statement of condition. For example, if volumes are expected to increase, the number of users served changed, or the activity deficiencies addressed, they should be expressed in the same terms as used in documenting the statement of condition.

Table 5-3, "Test Problem Documentation Example" illustrates what is normally documented to describe the problem and document the statement of condition and the statement of criteria. Note that an additional item could be added to describe the deviation. However, if the statement of condition and statement criteria is properly worded, the deviation should be readily determinable.

Table 5-3: Test Problem Documentation Example

Name of Software Tested	Put the name of the software system or subsystem tested here.
Problem Description	Write a brief narrative description of the variance uncovered from expectations.
Statement of Conditions	Put the results of actual processing that occurred here.
Statement of Criteria	Put what the testers believe was the expected result from processing.
Effect of Deviation	If this can be estimated, testers should indicate what they believe the impact or effect of the problem will be on computer processing.
Cause of Problem	The testers should indicate what they believe is the cause of the problem, if known. If the testers are unable to do this, the worksheet will be given to the development team and they should indicate the cause of the problem.
Location of Problem	The testers should document where the problem occurred as closely as possible. It can be related to a specific instruction or processing section that is desirable. If not, the testers should try to find the location as accurately as possible.
Recommended Action	The testers should indicate any recommended action they believe would be helpful to the project team. If the testers feel unable to indicate the action needed, the project team would record the recommended action here. Once approved, then the action would be implemented. If not approved, an alternate action should be listed or the reason for not following the recommended action should be documented.

5.4.2 Problem Effect

Whereas the legitimacy of a problem statement may stand or fall on criteria, the attention that the problem statement gets after it is reported depends largely on its significance. Significance is judged by effect.

Efficiency, economy, and effectiveness are useful measures of effect and frequently can be stated in quantitative terms such as dollars, time, and units of production, number of procedures and processes, or transactions. Where past effects cannot be ascertained, potential future effects may be presented. Sometimes, effects are intangible, but nevertheless of major significance.

In thought processes, effect is frequently considered almost simultaneously with the first two attributes of the problem. Reviewers may suspect a bad effect even before they have clearly formulated these other attributes in their minds. After the statement of condition is identified the reviewer may search for a firm criterion against which to measure the suspected effect. They may hypothesize several alternative criteria, which are believed to be suitable based on experiences in similar situations elsewhere. They may conclude that the effects under each

hypothesis are so divergent or unreasonable that what is really needed is a more firm criterion – say, a formal policy in an area where no policy presently exists. The presentation of the problem statement may revolve around this missing criterion, although suspicions as to effect may have been the initial path.

The reviewer should attempt to quantify the effect of a problem wherever practical. While the effect can be stated in narrative or qualitative terms, that frequently does not convey the appropriate message to management; for example, statements like "Service will be delayed," or "Extra computer time will be required" do not really tell what is happening to the organization.

5.4.3 Problem Cause

The cause is the underlying reason for the condition. In some cases the cause may be obvious from the facts presented. In other instances investigation will need to be undertaken to identify the origin of the problem.

Most findings involve one or more of the following causes:

- Nonconformity with standards, procedures, or guidelines
- Nonconformity with published instructions, directives, policies, or procedures from a higher authority
- Nonconformity with business practices generally accepted as sound
- Employment of inefficient or uneconomical practices

The determination of the cause of a condition usually requires the scientific approach, which encompasses the following steps:

Step 1. Define the problem (the condition that results in the finding).

Step 2. Identify the flow of work and information leading to the condition.

Step 3. Identify the procedures used in producing the condition.

Step 4. Identify the people involved.

Step 5. Recreate the circumstances to identify the cause of a condition.

5.4.4 Use of Test Results

Decisions need to be made as to who should receive the results of testing. Obviously, the developers whose products have been tested are the primary recipients of the results of testing. However, other stakeholders have an interest in the results including:

- End users
- Software project manager
- IT quality assurance

It is important to note that the individual whose results are being reported receive those results prior to other parties. This has two advantages for the software tester. The first is that the individual, whom testers believe may have made a defect, will have the opportunity to confirm or reject that defect. Second it is important for building good relationships between testers and developers to inform the developer who made the defect prior to submitting the data to other parties. Should the other parties contact the developer in question prior to the developer receiving the information from the tester, the developer would be put in a difficult situation. It would also impair the developer-tester relationship.

5.5 Defect Management

A major test objective is to identify defects. Once identified, defects need to be recorded and tracked until appropriate action is taken. This section explains a philosophy and a process to find defects as quickly as possible and minimize their impact.

This section also outlines an approach for defect management. This approach is a synthesis of the best IT practices for defect management. It is way to explain a defect management process within an organization.

Although the tester may not be responsible for the entire defect management process, they need to understand all aspects of defect management. The defect management process involves these general principles:

- The primary goal is to prevent defects. Where this is not possible or practical, the goals are to both find the defect as quickly as possible and minimize the impact of the defect.
- The defect management process, like the entire software development process, should be risk driven. Strategies, priorities and resources should be based on an assessment of the risk and the degree to which the expected impact of a risk can be reduced.
- Defect measurement should be integrated into the development process and be used by the project team to improve the development process. In other words, information on defects should be captured at the source as a natural by-product of doing the job. It should not be done after the fact by people unrelated to the project or system.
- As much as possible, the capture and analysis of the information should be automated. This section includes a list of tools which have defect management capabilities and can be used to automate some of the defect management processes.
- Defect information should be used to improve the process. This, in fact, is the primary reason for gathering defect information.
- Imperfect or flawed processes cause most defects. Thus, to prevent defects, the process must be altered.

5.5.1 Defect Naming Guidelines

It is important to name defects early in the defect management process. This will enable individuals to begin articulating more specifically what the defect is.

5.5.1.1 Name of the Defect

Name defects according to the phase in which the defect most likely occurred such as, requirements defect, design defect, documentation defect, and so forth.

5.5.1.2 Defect Severity

Use three categories of severity as follows:

- **Critical** - The defect(s) would stop the software system from operating.
- **Major** - The defect(s) would cause incorrect output to be produced.
- **Minor** - The defect(s) would be a problem but would not cause improper output to be produced, such as a system documentation error.

5.5.1.3 Defect Type

Indicates the cause of the defect. For example, code defects could be errors in procedural logic, or code that does not satisfy requirements or deviates from standards.

5.5.1.4 Defect Class

The following defect categories are suggested for each phase:

- **Missing** - A specification was not included in the software.
- **Wrong** - A specification was improperly implemented in the software.
- **Extra** - An element in the software was not requested by a specification

5.5.1.4.1 Defect-Naming Example

If a requirement was not correct because it had not been described completely during the requirements phase of development, the name of that defect using all 3 levels might be:

- **Name** – Requirement defect
- **Severity** – Minor
- **Type** - Procedural
- **Class** – Missing

Note that Severity, Type, and Class are qualifiers to the name.

5.5.2 The Defect Management Process

Figure 5-3 illustrates the key elements of a defect management process. Each element is described below.

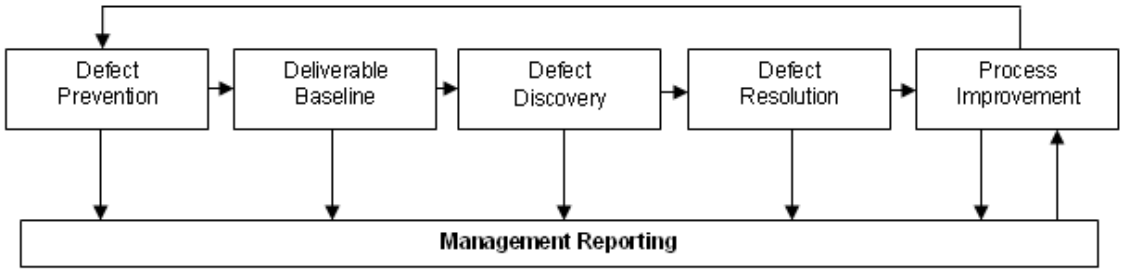


Figure 5-3 Defect Management Process

5.5.2.1 Defect Prevention

As many quality experts have pointed out, the best approach to defects is to eliminate them altogether. Until the technology exists to guarantee that defects will not be created, strategies will be needed to find them as quickly as possible and minimize their impact. Nevertheless, there is much that organizations can do to prevent defects. Identifying the best defect-prevention techniques (which is a large part of identifying the best software development processes) and implementing them should be a high-priority activity in any defect management program.

Figure 5-4 illustrates a defect prevention process with three major steps that are described below.

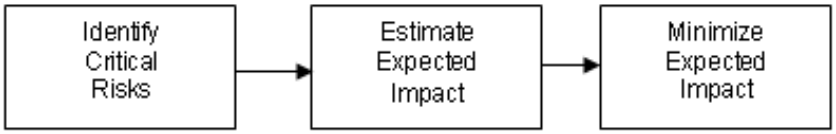


Figure 5-4 Defect Prevention

Defect prevention should begin with an assessment of the critical risks associated with the system. Once the critical risks are identified, it is possible to know the types of defects that are most likely to occur and the types of defects that can have the greatest impact on the system. Strategies can then be developed to prevent the defects.

Identify Critical Risks

The first step in preventing defects is to understand the critical risks facing the project or system. The best way to do this is to identify the types of defects that pose the

largest threat – defects that could jeopardize the successful construction, delivery, and operation of the system. These risks can vary widely from project to project depending on the type of system, the technology, the users of the software, etc. These risks might include:

- A key requirement is missing
- Critical application software does not function properly
- Vendor-supplied software does not function properly
- Software does not support major business functions – necessitates process re-engineering
- Performance is unacceptably poor
- Hardware that malfunctions
- Hardware and software do not integrate properly
- Hardware that is new to installation site
- Users are unable or unwilling to embrace new system
- User's ability to actively participate in project, etc.

It should be emphasized that the purpose of this step is not to identify every conceivable risk, rather to identify those critical risks, which could jeopardize the success of the project and therefore merit special attention

Estimate Expected Impact

The "expected impact" of a risk is affected by both the probability that the risk will become a problem and the potential loss to the organization. Estimating the expected impact provides insight into the issues involved in reducing the risk. For each critical risk, an assessment can be made of the impact, in dollars, if the risk does become a problem. The probability that the risk will become a problem can also be assessed. The product of these two numbers is the expected impact. Risks should be prioritized by the expected impact and the degree to which the expected impact can be reduced. While there will almost surely be a significant amount of guesswork in producing these numbers, precision is not important. What will be important is to identify the risk, and determine the order of magnitude of the risk.

Large, complex systems will have many critical risks and it will be important to reduce the probability of each individual critical risk becoming a problem to a very small number. In this circumstance the cumulative probability that one or more critical risks will become a problem is essentially the probability that the project will be successful. One should assume that an individual critical risk has a low probability of becoming a problem only when there is specific knowledge justifying why it is low. For example, the likelihood that an important requirement was missed may be high if users have not been very involved in the project. It may be low if the users have actively participated in the requirements definition using a good verification or validation process, and the new system is not a radical departure from an existing system or process.

One of the more effective methods for estimating the expected impact of a risk is the annual loss expectation formula. This formula states that the annual loss expectation

(ALE) equals the loss per event multiplied by the number of events. For the annual calculation, the number of events should be the number of events per year. The estimated loss can be calculated by determining the average loss for a sample of loss events. For example, if the risk is that the software system will abnormally terminate, then the average cost of correcting an abnormal termination is calculated and multiplied by the expected number of abnormal terminations associated with this risk.

The expected impact may be strongly affected not only by whether or not a risk becomes a problem, but also by how long it takes a problem to become recognized and how long it takes to be fixed, once recognized. In one reported example, a telephone company had an error in its billing system which caused it to under bill its customers by about $30 million. By law, the telephone company had to issue corrected bills within thirty days, or write off the under billing. By the time the telephone company recognized it had a problem, it was too late to collect much of the revenue.

Minimize Expected Impact

Expected impact is also affected by the action that is taken once a problem is recognized. Once Johnson & Johnson realized it had a problem with Tylenol tampering, it greatly reduced the impact of the problem by quickly notifying doctors, hospitals, distributors, retail outlets, and the public, of the problem. While the tampering itself was not related to a software defect, software systems had been developed by Johnson & Johnson to quickly respond to drug-related problems. In this case, the key to Johnson & Johnson's successful management of the problem was how it minimized the impact of the problem, once the problem was discovered.

Minimizing expected impact involves a combination of the following three strategies:

- *Eliminate the risk*. While this is not always possible, there are situations where the best strategy will be simply to avoid the risk. For example, reducing the scope of a system, or deciding not to use the latest unproven technology, are ways to avoid certain risks.
- *Reduce the probability of a risk becoming a problem*. Most strategies will fall into this category. Inspections and testing are examples of approaches, which reduce, but do not eliminate, the probability of problems.
- *Reduce the impact if there is a problem.* In some situations, the risk cannot be eliminated, and even when the probability of a problem is low, the expected impact is high. In these cases, the best strategy may be to explore ways to reduce the impact if there is a problem. Contingency plans and disaster recovery plans would be examples of this strategy.

From a conceptual viewpoint, there are two ways to minimize the risk. These are deduced from the annual loss expectation formula. The two ways are:

- Reduce the expected loss per event
- Reduce the frequency of an event

If both of these can be reduced to zero, the risk will be eliminated. If the frequency is reduced, the probability of a risk becoming a problem is reduced. If the loss per event is reduced, the impact is reduced when the problem occurs.

There is a well-known engineering principle that says that if you have a machine with a large number of components, even if the probability that any given component will fail is small, the probability that one or more components will fail may be unacceptably high. Because of this phenomenon, engineers are careful to estimate the mean time between failures of the machine. If the machine cannot be designed with a sufficiently large mean time between failures, the machine cannot be made. When applied to software development, this principle would say that unless the overall expected impact of the system can be made sufficiently low, do not develop the system.

Appropriate techniques to reduce expected impact are a function of the particular risk. Techniques to prevent defects include:

- Quality Assurance

  Quality assurance techniques are designed to ensure that the processes employed are adequate to produce the desired result and that the process is being followed.
- Training and Education (Work Force)

  It goes without saying that the better trained a work force is, the higher the quality of its work. Many defects are simply the result of workers not understanding how to do their job. Computer technology is significantly more complex today than it was just a few years ago. Moreover, the complexity will increase significantly in the coming years. Thus, it appears that the training needs at most organizations will increase sharply in the coming years.
- Training and Education (Customers)

  As more and more people use computers, and as the complexity of systems grows, the problem of training the end user will become more and more acute. Unlike the problem of training workers, more creative strategies will be required to train customers – especially when customers are not technically sophisticated. One computer manufacturer reported that a customer, when asked to send in a copy of a disk, sent in a Xerox copy of the disk. In another instance, a customer complained that a disk would not go into the drive. It was later determined that the customer did not realize that he had to remove the disk that was already in the drive before another could be inserted. While these anecdotes are extreme, the problem of effectively training even sophisticated customers to use complex software is far from trivial. Many software vendors have recognized this problem and developed strategies to address it (more elaborate Help facilities, cue cards, audio training tapes delivered with the product, tutorials, etc.).
- Methodology and Standards

  As Deming emphasizes, reducing variation is key to ensuring quality. As the nature of a process becomes understood, it evolves from art to science. At some point in this evolution, it becomes appropriate to standardize the process. This has occurred in the past with the development of standard life cycles, design

approaches, etc. This is occurring today with many diverse efforts – various IEEE standards, ISO 9000, etc. As the root cause of defects becomes understood, consideration should be given to developing or enhancing an organization's methodology and standards to produce a repeatable process that prevents the defects from reoccurring.

- Defensive Design

  While there are many variations of defensive design, the concept generally refers to designing the system so that two or more independent parts of the system must fail before a failure could occur. As technology gets more and more complicated, there should be significantly more emphasis on designing systems defensively to prevent, discover, and minimize the impact of defects. While some organizations have been doing this for years, it is a new concept to many organizations and the industry provides very little guidance on how to do it. Design techniques to improve reliability should receive more attention as the complexity of technology grows. These techniques usually involve designing the system so that two components of the system must be in error before a major system problem can occur.

- Defensive Code

  The concept of designing a program to prevent, discover, and minimize the impact of defects is not new; it is, however, not widely practiced. Like defensive design, the concept of defensive code involves adding code to a program so that two parts of the program must fail before a major problem can occur. One form of defensive design, assertions, has been around for many years, but has received relatively little attention. An assertion is a code which tests for expected conditions and brings unexpected conditions to the attention of the programmer or users. This area also deserves to receive more attention as the complexity of technology grows.

  The best defect-prevention techniques will be the ones, which reduce the expected impact the most. This, in turn, will be a function of the nature of the risks and systems within an organization. Very critical software (e.g., NASA's space shuttle software, health care equipment) and widely distributed software (e.g., Microsoft Windows) may need to use all of the above techniques and more to adequately reduce the overall risk of highly critical software.

5.5.2.2 Deliverable Baseline

You baseline a deliverable, or work product when it reaches a predefined milestone in its development. This milestone involves transferring the product from one stage of development to the next. As a work product moves from one milestone to the next, the cost to make changes becomes much more expensive and defects in the deliverable have a much larger impact on the rest of the system. Once you baseline a deliverable, it is subject to configuration management (e.g., change control).

A defect is an instance of one or more baseline product components not satisfying their given set of requirements. Thus, errors caught before a deliverable is baselined are not to be considered defects. For example, if a programmer had responsibility for both the

programming and the unit testing of a module, the program would not become baselined until after the program was unit tested. Therefore, a potential defect discovered during unit testing is not considered a defect. If, on the other hand, an organization decided to separate the coding and unit testing, it might decide to baseline the program after it was coded, but before it was unit tested. In this case, a potential defect discovered during unit testing would be considered a defect as illustrated in Figure 5-5 Deliverable Baseline.

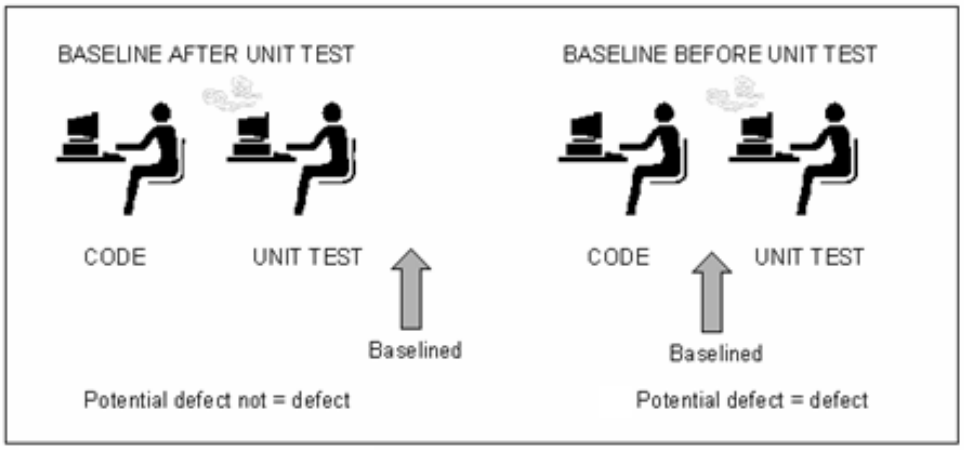


Figure 5-5 Deliverable Baseline

Even if an organization does not formally recognize the concept of baselining deliverables, a deliverable is, for practical purposes, baselined when the person or group responsible for the deliverable passes it to the next stage of development. For example, a program specification should be considered baselined when a programmer is using it as the basis to code a program; a program should be considered baselined when it is passed on for integration testing; and requirements specification should be considered baselined if it is being used as the basis for a technical design.

The concept of baselining is important because it requires an organization to decide both the level of formality that is appropriate and the point in the process when the formality takes effect. In general, a deliverable should be baselined when changes to the deliverable, or defects in the deliverable, can have an impact on deliverables on which other people are working.

Deliverable baseline involves the following activities:

- Identify key deliverables

  Select those deliverables which will be baselined, and the point within the development process where those deliverable will be baselined.

- Define standards for each deliverable

  The standards should define the requirements for each deliverable and the criteria that must be met before the deliverable can be baselined.

5.5.2.3 Defect Discovery

If technology cannot guarantee that defects will not be created, and this is certainly the case in software development today, then the next best thing is to find defects quickly before the cost to fix them is great. A defect is considered to have been discovered when the defect has been formally brought to the attention of the developers, and the developers acknowledge that the defect is valid. A defect has not necessarily been discovered when the user finds a problem with the software. The user must also report the defect and the developers must acknowledge that the defect is valid. There are examples where users reported problems for years before the developers of the software admitted there was a defect. Since it is important to minimize the time between defect origination and defect discovery, strategies that not only uncover the defect, but also facilitate the reporting and developer acknowledgment of the defect can be very important.

To make it easier to recognize defects, organizations should attempt to predefine defects by category. This is a one-time event, or an event that could be performed annually. It would involve the knowledgeable, respected individuals from all major areas of the IT organization. A facilitator should run the group. The objective is to identify the errors or problems that occur most frequently in the IT organization and then get agreement that they are, in fact, defects. A name should be attached to the defects. The objective of this activity is to avoid conflict when developers do not acknowledge identified defects as a valid defect. For example, developers may not want to acknowledge that a missing requirement is a defect, but if it has previously been defined as a defect, the developer's concurrence is not necessary.

The steps involved in defect discovery are illustrated in Figure 5-6 and described below.

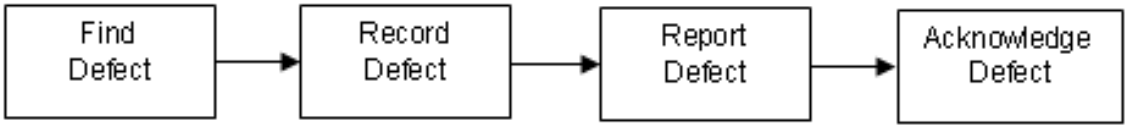


Figure 5-6 Defect Discovery

- Find Defect

Defects are found either by pre-planned activities specifically intended to uncover defects (e.g., quality control activities such as inspections, testing, etc.) or, in effect, by accident (e.g., users in production).

Techniques to find defects can be divided into three categories:

- Static techniques – A deliverable is examined (manually or by a tool) for defects. Reviews, walkthroughs, and inspections, are examples of static techniques.
- Dynamic techniques – A deliverable is used to discover defects. Testing is an example of a dynamic technique.
- Operational techniques – An operational system produces a deliverable containing a defect found by users, customers, or control personnel, i.e., the defect is found as a result of a failure.

Research shows the following conclusions when you compare and contrast the various static, dynamic, and operational techniques.

- Each of the three categories of techniques is generally required for an effective defect management program. In each category, the more formally the techniques were integrated into the development process, the more effective they were.
- Since static techniques will generally find defects earlier in the process, they are more efficient at finding defects.
- Inspection, in particular, can be very effective at removing defects. NASA and Shell Oil both use them extensively and have had impressive results. When the full inspections process was followed, Shell Oil found that for each staff-hour spent in the inspection process, ten hours were saved! Shell Oil also found that even when groups decided not to follow the prescribed inspections process, but conducted more informal (and less effective) reviews, the results showed that the reviews saved as much time as they cost. In other words, worst case (informal reviews) – no extra cost; best case (formal inspections) – a 10-1 savings. Shell Oil found that their defect removal efficiency with inspections was 95-97% versus roughly 60% for systems that did not use inspections.

Shell Oil also emphasized the more intangible, yet very significant, benefits of inspections:

- If the standards for producing a deliverable were vague or ambiguous (or nonexistent), the group would attempt to define a best practice and develop a standard for the deliverable. Once the standard became well defined, checklists would be developed. (NASA also makes extensive use of checklists and cross-references defects to the checklist item that should have caught the defect).
- Inspections were a good way to train new staff in both, best practices and the functioning of the system being inspected.

- Record Defect

  Recording the defects identified at each stage of the test process is an integral part of a successful life cycle testing approach. The purpose of this activity is to create a complete record of the discrepancies identified during testing. The information captured is used in multiple ways throughout the project, and forms the basis for quality measurement.

A defect can be defined in one of two ways. From the producer's viewpoint, a defect is a deviation from specifications, whether missing, wrong, or extra. From the Customer's viewpoint, a defect is anything that causes customer dissatisfaction, whether in the requirements or not; this view is known as "fit for use." It is critical that defects identified at each stage of the project life cycle be tracked to resolution.

You should record defects for these four major purposes:

- To correct the defect
- To report status of the application
- To gather statistics used to develop defect expectations in future applications
- To improve the software development process

Most project teams utilize some type of tool to support the defect tracking process. This tool could be as simple as a white board or a table created and maintained in a word processor, or one of the more robust tools available today on the market, such as Mercury's Test Director. Tools marketed for this purpose usually come with some number of customizable fields for tracking project specific data in addition to the basics. They also provide advanced features such as standard and ad hoc reporting, e-mail notification to developers and testers when a problem is assigned to them, and graphing capabilities.

At a minimum, the tool selected should support the recording and communication of all significant information about a defect. For example, a defect log could include:

- Defect ID number
- Descriptive defect name and type
- Source of defect – test case or other source
- Defect severity
- Defect priority
- Defect status (e.g., open, fixed, closed, user error, design, and so on) – more robust tools provide a status history for the defect
- Date and time tracking for either the most recent status change, or for each change in the status history
- Detailed description, including the steps necessary to reproduce the defect
- Component or program where defect was found
- Screen prints, logs, etc., that will aid the developer in the resolution process
- Stage of origination
- Person assigned to research and correct the defect

5.5.2.3.1 Severity versus Priority

The test team, based on pre-defined severity descriptions, should assign the severity of a defect objectively. For example a "severity one" defect may be defined as one that causes data corruption, a system crash, security violations, etc. In large projects, it may also be necessary to assign a priority to the defect, which determines the order in which defects should be fixed. The priority assigned to a defect is usually more subjective based upon input from users regarding which defects are most important to them, and therefore should be fixed first.

It is recommended that severity levels be defined at the start of the project so that they are consistently assigned and understood by the team. This foresight can help test teams avoid the common disagreements with development teams about the criticality of a defect.

5.5.2.3.2 A Sample Defect-Tracking Process

After a defect is recorded it needs to be tracked. The steps below describe a simple defect tracking process. Depending on the size of the project or project team, this process may be substantially more complex.

1. Execute the test and compare the actual results to the documented expected results. If a discrepancy exists, log the discrepancy with a status of "open." Supplementary documentation, such as screen prints or program traces, should be attached if available.
2. The Test Manager or tester should review the problem log with the appropriate member of the development team to determine if the discrepancy is truly a defect.
3. Assign the defect to a developer for correction. Once the defect is corrected, the developer will usually enter a description of the fix applied and update the defect status to "Fixed" or "Retest."
4. The defect is routed back to the test team for retesting. Additional regression testing is performed as needed based on the severity and impact of the fix applied.
5. If the retest results match the expected results, the defect status is updated to "Closed." If the test results indicate that the defect is still not fixed, the status is changed to "Open" and sent back to the developer.

Steps 3-5 should be repeated until the problem is resolved. Test reports are issued periodically throughout the testing process to communicate the test status to the rest of the team and management. These reports usually include a summary of the open defects, by severity. Additional graphs and metrics can also be provided to further describe the status of the application.

5.5.2.3.3 Report Defects

Once found, defects must be brought to the attention of the developers. When the defect is found by a technique specifically designed to find defects, such as those mentioned above, this is a relatively straightforward process and is almost as simple as writing a problem report. Some defects, however, are found more by accident – people who are not trying to find defects. These may be development personnel or users. In these cases, techniques that facilitate the reporting of the defect may significantly shorten the defect discovery time. As software becomes more complex and more widely used, these techniques become more valuable. These techniques include computer forums, electronic mail, help desks, etc.

It should also be noted that there are some human factors and cultural issues involved with the defect discovery process. When a defect is initially uncovered, it may be very unclear whether it is a defect, a change, user error, or a misunderstanding. Developers may resist calling something a defect because that implies "bad work" and may not reflect well on the development team. Users may resist calling something a "change" because that implies that the developers can charge them more money. Some organizations have skirted this issue by initially labeling everything by a different name – for example, "incidents" or "issues." From a defect management perspective, what they are called is not an important issue. What is important is that the defect be quickly brought to the developers' attention and formally controlled.

5.5.2.3.4 Acknowledge Defect

Once a defect has been brought to the attention of the developer, the developer must decide whether or not the defect is valid. Delays in acknowledging defects can be very costly. The primary causes of delays in acknowledging a defect appears to be an inability to reproduce the defect. When the defect is not reproducible and appears to be an isolated event ("no one else has reported anything like that"), there will be an increased tendency for the developer to assume the defect is not valid – that the defect is caused by user error or misunderstanding. Moreover, with very little information to go on, the developer may feel that there is nothing they can do anyway. Unfortunately, as technology becomes more complex, defects, which are difficult to reproduce, will become more and more common. Software developers must develop strategies to quickly pinpoint the cause of a defect.

Strategies to address this problem include:

- Instrument the code to trap the state of the environment when anomalous conditions occur.

  In the Beta release of Windows 3.1, Microsoft included features to trap the state of the system when a significant problem occurred. This information was then available to Microsoft when the problem was reported and helped them analyze the problem.

- Write code to check the validity of the system.

  This is actually a very common technique for hardware manufacturers. Unfortunately, diagnostics may give a false sense of security – they can find defects, but they cannot show the absence of defects. Virus checkers would be an example of this strategy.

- Analyze reported defects to discover the cause of a defect.

  While a given defect may not be reproducible, quite often it will appear again (and again) perhaps in different disguises. Eventually patterns may be noticed, which will help in resolving the defect. If the defect is not logged, or if it is closed prematurely, then valuable information can be lost. In one instance reported to the research team, a development team was having difficulty reproducing a problem. They noticed, however, that the defect was showing up at only one location. Finally, during a visit to the location they discovered how to reproduce the problem. The problem was caused when one of the users fell asleep with her finger on the enter key. In order to protect the user, the circumstances surrounding the problem were not reported to the developers until the on-site visit.

A resolution process needs to be established for use in the event there is a dispute regarding a defect. For example, if the group uncovering the defect believes it is a defect but the developers do not, a quick-resolution process must be in place. While many approaches can address this situation, the two most effective are:

- Arbitration by the software owner – the customer of the software determines whether or not the problem shall be called a defect.
- Arbitration by a software development manager – a senior manager of the software development department will be selected to resolve the dispute.

5.5.2.4 Defect Resolution

Once the developers have acknowledged that a reported defect is a valid defect, the defect resolution process begins. The steps involved in defect resolution are illustrated in Figure 5-7 and described below.

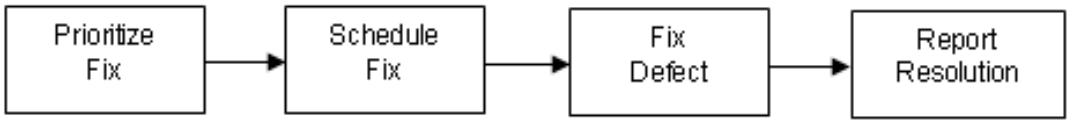


Figure 5-7 Defect Resolution

5.5.2.4.1 Prioritize Fix

The purpose of this step is to answer the following questions and initiate any immediate action that might be required:

- Is this a previously reported defect, or is it new?
- What priority should be given to fixing this defect?
- What steps should be taken to minimize the impact of the defect prior to a fix? For example, should other users be notified of the problem? Is there a work-around for the defect?

This page intentionally left blank.

A suggested prioritization method is a three-level method, as follows:

- Critical – Would have a serious impact on the organization's business operation.
- Major – Would cause an output of the software to be incorrect or stop.
- Minor – Something is wrong, but it does not directly affect the user of the system, such as a documentation error or cosmetic GUI (graphical user interface) error.

5.5.2.4.2 Schedule Fix

Based on the priority of the defect, the fix should be scheduled. It should be noted that some organizations treat lower priority defects as changes. All defects are not created equal from the perspective of how quickly they need to be fixed. (From a defect-prevention perspective, they may all be equal.)

5.5.2.4.3 Fix Defect

This step involves correcting and verifying one or more deliverables (e.g., programs, documentation) required to remove the defect from the system. In addition, test data, checklists, etc., should be reviewed and perhaps enhanced, so that, in the future, this defect would be caught earlier.

5.5.2.4.4 Report Resolution

Once the defect has been fixed and the fix verified, appropriate developers and users need to be notified that the defect has been fixed, the nature of the fix, when the fix will be released, and how the fix will be released. As in many aspects of defect management, this is an area where automation of the process can help. Most defect management tools capture information on who found and reported the problem and therefore provides an initial list of who needs to be notified. Figure 5-8 illustrates this process. Computer forums and electronic mail can help notify users of widely distributed software.

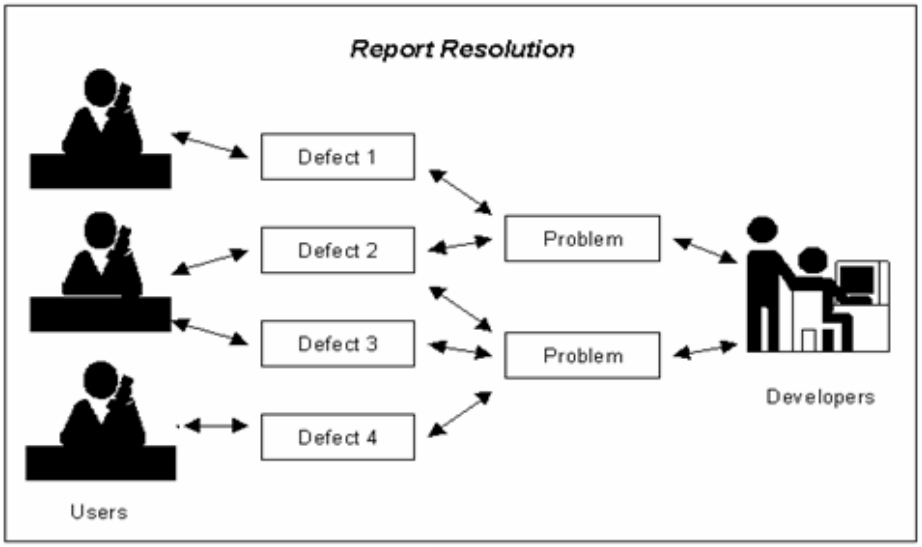


Figure 5-8 Report Resolution

5.5.2.5 Process Improvement

This is perhaps the activity that is most ignored by organizations today, but offers one of the greatest areas of payback. NASA emphasizes the point that any defect represents a weakness in the process. Seemingly unimportant defects are, from a process perspective, no different than critical defects. It is only the developer's good luck that prevents a defect from causing a major failure. Even minor defects, therefore, represent an opportunity to learn how to improve the process and prevent potentially major failures. While the defect itself may not be a big deal, the fact that there was a defect is a big deal.

This activity should include the following:

- Go back to the process where the defect originated to understand what caused the defect.
- Go back to the validation process, which should have caught the defect earlier in the process. Not only can valuable insight be gained as to how to strengthen the review process, this step serves to make everyone involved in these activities take them more seriously. This human factors dimension alone, according to some of the people the

research team interviewed, can have a very large impact on the effectiveness of the review process.

NASA takes an additional step of asking the question: If this defect could have gotten this far into the process before it was captured, what other defects may be present that have not been discovered? Thus, not only is the process strengthened to prevent defects, it is strengthened to find defects which have been created but not yet discovered. This aggressiveness should be mandatory on life-critical systems.

Skill Category

6

Test Reporting Process

Testers need to demonstrate the ability to develop testing status reports. These reports should show the status of the testing based on the test plan. Reporting should document what tests have been performed and the status of those tests. The test reporting process is a process to collect data, analyze the data, supplement the data with metrics, graphs and charts and other pictorial representations which help the developers and users interpret that data. The lessons learned from the test effort should be used to improve the next iteration of the test process.

Prerequisites to Test Reporting	**6-1**
Test Tools used to Build Test Reports	**6-11**
Test Tools used to Enhance Test Reporting	**6-32**
Reporting Test Results	**6-37**

6.1 Prerequisites to Test Reporting

From the developer and user perspective the value of software testing is in the reports issued by the testers. The testers uncover facts, document those facts into a finding, and then report that information to developers and users. They may also provide opinions and recommendations under findings. The test reporting process begins with the prerequisite to collect test status data, analyze the data, and supplement the data with effective metrics.

It is recommended that a database be established in which to store the results collected during testing. It is also suggested that the database be put online through client/server systems so that those with a vested interest in the status of the project can readily access that database for status update.

The prerequisites to the process of reporting test results are:

- Define the test status data to be collected
- Define the test metrics to be used in reporting test results
- Define effective test metrics

6.1.1 Define and Collect Test Status Data

Processes need to be put into place to collect the data on the status of testing that will be used in reporting test results. Before these processes are built testers need to define the data they need to collect. Four categories of data that testers most frequently collect are:

- Test results data
- Test case results and test verification results
- Defects
- Efficiency

6.1.1.1 Test Results Data

This data will include, but is not limited to:

- Test factors – The factors incorporated in the plan, the validation of which becomes the test objective.
- Business objectives – The validation that specific business objectives have been met.
- Interface objectives – Validation that data/objects can be correctly passed among software components.
- Functions and sub-functions – Identifiable software components normally associated with the requirements for the software.
- Units – The smallest identifiable software components.
- Platform – The hardware and software environment in which the software system will operate.

6.1.1.2 Test Case Results and Test Verification Results

These are the test techniques used by the test team to perform testing. They include, but are not limited to:

- Test cases – The type of tests that will be conducted during the execution of tests, which will be based on software requirements.
- Inspections – A verification of process deliverables against deliverable specifications.
- Reviews – Verification that the process deliverables/phases are meeting the user's true needs.

6.1.1.3 Defects

This category includes a description of the individual defects uncovered during testing. The description of defects should include, but is not limited to:

- Data the defect uncovered
- Name of the defect
- Location of the defect
- Severity of the defect
- Type of defect
- How the defect was uncovered (i.e., test data/test script)

The results of later investigations should add to this information and include:

- where the defect originated
- when it was corrected
- when it was entered for retest

6.1.1.4 Efficiency

As the Test Plan is being developed, the testers decompose requirements into lower and lower levels. Conducting testing is normally a reverse of the test planning process. In other words, testing begins at the very lowest level and the results are rolled up to the highest level. The final Test Report determines whether the requirements were met. How well documenting, analyzing, and rolling up test results proceeds depends partially on the process of decomposing testing through a detailed level. The roll-up is the exact reverse of the test strategy and tactics. The efficiency of these processes should be measured.

Two types of efficiency can be evaluated during testing: efficiency of the software system and efficiency of the test process. If included in the mission of software testing, the testers can measure the efficiency of both developing and operating the software system. This can involve simple metrics such as the cost to produce a function point of logic, or as complex as using measurement software.

6.1.2 Define Test Metrics used in Reporting

The most common Test Report is a simple matrix, which indicates the test cases, the test objectives, and the results of testing at any point in time.

The following six tasks define how a test team can define metrics to be used in test reporting. An important part of these tasks is to assure that the data needed (i.e., measures) to create the test metrics is available.

1. Establish a test metrics team.

   The measurement team should include individuals who:

- Have a working knowledge of quality and productivity measures
- Are knowledgeable in the implementation of statistical process control tools
- Have a working understanding of benchmarking techniques
- Know the organization's goals and objectives
- Are respected by their peers and management

The measurement team may consist of two or more individuals, relative to the size of the organization. Representatives should come from management and development and maintenance projects. For an average-size organization, the measurement team should be between three and five members.

2. Inventory existing IT measures.

   The inventory of existing measures should be performed in accordance with a plan. Should problems arise during the inventory, the plan and the inventory process should be modified accordingly. The formal inventory is a systematic and independent review of all existing measures and metrics captured and maintained. All identified data must be validated to determine if they are valid and reliable.

   The inventory process should start with an introductory meeting of the participants. The objective of this meeting is to review the inventory plan with management and representatives of the projects that are to be inventoried. A sample agenda for the introductory meeting is:

   - Introduce all members.
   - Review scope and objectives of the inventory process
   - Summarize the inventory processes to be used
   - Establish communication channels to use
   - Confirm the inventory schedule with major target dates

     The inventory involves these activities:

     - *Review all measures currently being captured and recorded.* Measures should include, but not be limited to, functionality, schedule, budgets, and quality.

     - *Document all findings.* Measures should be defined; samples captured, and related software and methods of capture documented. Data file names and media location should be recorded. It is critical that this be as complete as possible in order to determine the consistency of activities among different projects.

     - *Conduct interview*s. These interviews should determine what and how measurement data is captured and processed. Through observation, the validity of the data can be determined.

3. Develop a consistent set of metrics.

   To implement a common set of test metrics for reporting that enables senior management to quickly access the status of each project, it is critical to develop a list of consistent measures spanning all project lines. Initially, this can be challenging, but with cooperation

and some negotiating, a reasonable list of measures can be drawn up. Organizations with mature processes will have an easier time completing this step, as well as those with automated tools that collect data.

4. Define desired test metrics.

   The objective of this task is to use the information collected in tasks 2 and 3 to define the metrics for the test reporting process. Major criteria of this task includes:

   - Description of desired output reports
   - Description of common measures
   - Source of common measures and associated software tools for capture
   - Definition of data repositories (centralized and/or segregated)

5. Develop and implement the process for collecting measurement data.

   The objective of this step is to document the process used to collect the measurement data. The implementation will involve these activities:

   - Document the workflow of the data capture and reporting process
   - Procure software tool(s) to capture, analyze, and report the data, if such tools are not currently available
   - Develop and test system and user documentation
   - Beta-test the process using a small to medium-size project
   - Resolve all management and project problems
   - Conduct training sessions for management and project personnel on how to use the process and interrelate the reports
   - Roll out the test status process

6. Monitor the process.

   Monitoring the test reporting process is very important because the metrics reported must be understood and used. It is essential to monitor the outputs of the system to ensure usage. The more successful the test reporting process, the better the chance that management will want to use it and perhaps expand the reporting criteria.

6.1.3 Define Effective Test Metrics

A metric is a mathematical number that shows a relationship between two variables. Software metrics are measures used to quantify status or results. This includes items that are directly measurable, such as lines of code, as well as items that are calculated from measurements, such as earned value. Metrics specific to testing include data regarding testing, defect tracking, and software performance. The following are metric definitions:

Metric

A metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.

Process Metric

A process metric is a metric used to measure characteristics of the methods, techniques, and tools employed in developing, implementing, and maintaining the software system.

Product Metric

A product metric is a metric used to measure the characteristics of the documentation and code.

Software Quality Metric

A software quality metric is a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.

Testers are typically responsible for reporting their test status at regular intervals. The following measurements generated during testing are applicable:

- Total number of tests
- Number of tests executed to date
- Number of tests executed successfully to date

Data concerning software defects include:

- Total number of defects corrected in each activity
- Total number of defects detected in each activity
- Average duration between defect detection and defect correction
- Average effort to correct a defect
- Total number of defects remaining at delivery

Some of the basic measurement concepts are described below to help testers use quantitative data effectively.

6.1.3.1 Objective versus Subjective Measures

Measurement can be either objective or subjective. An objective measure is a measure that can be obtained by counting. For example, objective data is hard data, such as defects, hours worked, and completed deliverables. Subjective data normally has to be calculated. It is a person's perception of a product or activity. For example, a subjective measure would involve such attributes of an information system as how easy it is to use and the skill level needed to execute the system.

As a general rule, subjective measures are much more important than objective measures. For example, it is more important to know how effective a person is in performing a job (a subjective measure) versus whether or not they got to work on time (an objective measure). QAI believes that the more difficult something is to measure, the more valuable that measure.

Individuals seem to want objective measures because they believe they are more reliable than subjective measures. It is unfortunate, but true, that many bosses are more concerned that the workers are at work on time and do not leave early, than they are about how productive they

are during the day. You may have observed the type of people that always want to arrive at work before the boss, because they believe meeting objective measures is more important than meeting subjective measures, such as how easy the systems they built are to use.

6.1.3.2 How Do You Know a Metric is Good?

Before a measure is approved for use, there are certain tests that it must pass. QAI has identified the following tests that each measure and metric should be subjected to before it is approved for use:

Reliability

This refers to the consistency of measurement. If taken by two people, would the same results be obtained?

Validity

This indicates the degree to which a measure actually measures what it was intended to measure.

Ease of Use and Simplicity

These are functions of how easy it is to capture and use the measurement data.

Timeliness

This refers to whether the data was reported in sufficient time to impact the decisions needed to manage effectively.

Calibration

This indicates the movement of a metric so it becomes more valid, for example, changing a customer survey so it better reflects the true opinions of the customer.

6.1.3.3 Standard Units of Measure

A measure is a single attribute of an entity. It is the basic building block for a measurement program. Measurement cannot be used effectively until the standard units of measure have been defined. You cannot intelligently talk about lines of code until the measure lines of code has been defined. For example, lines of code may mean lines of code written, executable lines of code written, or even non-compound lines of code written. If a line of code was written that contained a compound statement it would be counted as two or more lines of code, such as a nested IF statement two levels deep. In addition, organizations may desire to use weighting factors; for example, one verb would be weighted as more complete than other verbs in the same programming language.

Measurement programs can be started with as few as five or six standard units of measure, but rarely would exceed 50 standard units of measure.

6.1.3.4 Productivity versus Quality

Quality is an attribute of a product or service. Productivity is an attribute of a process. They have frequently been called two sides of the same coin. This is because one has a significant impact on the other.

There are two ways in which quality can drive productivity. The first, and an undesirable method, is to lower or not meet quality standards. For example, if one chose to eliminate the testing and rework components of a system development process, productivity as measured in lines of code per hours worked would be increased. This is done frequently in information services under the guise of completing projects on time. While testing and rework may not be eliminated, they are not complete when the project is placed into production. The second method for improving productivity through quality is to improve processes so that defects do not occur, thus minimizing the need for testing and rework. The QAI Approach uses quality improvement processes to drive productivity.

6.1.3.5 Test Metric Categories

While there are no generally accepted categories of metrics, it has proved helpful to many test organizations to establish categories for status and reporting purposes. The metric categories and metrics within those categories provide an inventory of the metrics that testers will use in status reporting and final test reports.

In examining many reports prepared by testers the following eight metric categories are commonly used:

- Metrics unique to test
- Complexity measurements
- Project metrics
- Size measurements
- Defect metrics
- Product measures
- Satisfaction metrics
- Productivity metrics

6.1.3.5.1 Metrics Unique to Test

This category includes metrics such as Defect Removal Efficiency, Defect Density, and Mean Time to Last Failure. The following are examples of metrics unique to test:

- Defect removal efficiency – the percentage of total defects occurring in a phase or activity removed by the end of that activity.
- Defect density – the number of defects in a particular product.
- Mean time to failure – the average operational time it takes before a software system fails.

- Mean time to last failure – an estimate of the time it will take to remove the last defect from the software
- Coverage metrics – the percentage of instructions or paths executed during tests.
- Test cycles – the number of testing cycles required to complete testing (Note: May be related to the size of the software system or complexity of the system).
- Requirements tested – the percentage of requirements tested during testing (Note: Can indicate requirements tested which are correct, and requirements tested having defects).

6.1.3.5.2 Complexity Measurements

This category includes quantitative values accumulated by a predetermined method, which measure the complexity of a software product. The following are examples of complexity measures:

- Size of module/unit (larger module/units are considered more complex).
- Logic complexity – the number of opportunities to branch/transfer within a single module.
- Documentation complexity – the difficulty level in reading documentation usually expressed as an academic grade level.

6.1.3.5.3 Project Metrics

This category includes status of the project including milestones, budget and schedule variance and project scope changes. The following are examples of project metrics:

- Percent of budget utilized
- Days behind or ahead of schedule
- Percent of change of project scope
- Percent of project completed (not a budget or schedule metric, but rather an assessment of the functionality/structure completed at a given point in time)

6.1.3.5.4 Size Measurements

This category includes methods primarily developed for measuring the software size of software systems, such as lines of code, and function points. These can also be used to measure software testing productivity. Sizing is important in normalizing data for comparison to other projects. The following are examples of size metrics:

- KLOC – thousand lines of code, used primarily with statement level languages.
- Function points – a defined unit of size for software.
- Pages or words of documentation

6.1.3.5.5 Defect Metrics

This category includes values associated with numbers or types of defects, usually related to system size, such as “defects/1000 lines of code” or “defects/100 function points,” severity of defects, uncorrected defects, etc. The following are examples of defect metrics:

- Defects related to size of software.
- Severity of defects such as very important, important, and unimportant.
- Priority of defects – the importance of correcting defects.
- Age of defects – the number of days the defect has been uncovered but not corrected.
- Defects uncovered in testing
- Cost to locate a defect

6.1.3.5.6 Product Measures

This category includes measures of a product’s attributes such as performance, reliability, usability. The following are examples of product measures:

- Defect density – the expected number of defects that will occur in a product during development.

6.1.3.5.7 Satisfaction Metrics

This category includes the assessment of customers of testing on the effectiveness and efficiency of testing. The following are examples of satisfaction metrics:

- Ease of use – the amount of effort required to use software and/or software documentation.
- Customer complaints – some relationship between customer complaints and size of system or number of transactions processed.
- Customer subjective assessment – a rating system that asks customers to rate their satisfaction on different project characteristics on a scale, for example a scale of 1-5.
- Acceptance criteria met – the number of user defined acceptance criteria met at the time software goes operational.
- User participation in software development – an indication of the user desire to produce high quality software on time and within budget.

6.1.3.5.8 Productivity Metrics

This category includes the effectiveness of test execution. Examples of productivity metrics are:

- Cost of testing in relation to overall project costs – assumes a commonly accepted ratio of the costs of development versus tests.
- Under budget/Ahead of schedule.
- Software defects uncovered after the software is placed into an operational status.
- Amount of testing using automated tools.

6.2 Test Tools used to Build Test Reports

Testers use many different tools to help in analyzing the results of testing, and to create the information contained in the test reports. The use of these tools has proven very effective in improving the value of the reports prepared by testers for the stakeholders of the software system.

Experience has shown the analysis and reporting of defects and other software attributes is enhanced when those involved are given analysis and reporting tools. Software quality professionals have recognized the following tools as the more important analysis tools used by software testers. Some of these tools are included in statistical software packages. For each tool the deployment, or how to use, is described, as well as examples, results, and recommendations.

6.2.1 Pareto Charts

A Pareto chart is a special type of bar chart to view the causes of a problem in order of severity: largest to smallest. The Pareto chart provides an effective tool to graphically show where significant problems and causes are in a process.

A Pareto chart can be used when data is available or can be readily collected from a process. The use of this tool occurs early in the continuous improvement process when there is a need to order or rank, by frequency of problems and causes. Team(s) can focus on the vital few problems and their respective root causes contributing to these problems. This technique provides the ability to:

- Categorize items, usually by content or cause factors.
  - Content: type of defect, place, position, process, time, etc.
  - Cause: materials, machinery or equipment, operating methods, manpower, measurements, etc.
- Identify the causes and characteristics that most contribute to a problem.
- Decide which problem to solve or basic causes of a problem to work on first.
- Understand the effectiveness of the improvement by doing pre- and post-improvement charts.

6.2.1.1 Deployment

A process for using Pareto charts requires a series of tasks, which fall into the following steps:

1. Define the problem clearly – Normally, this results from a team's brainstorming including using techniques such as development of affinity diagrams and cause and effect (fishbone) diagrams.
2. Collect data – Sufficient sample size over specified time, or use historical data if available or retrievable.

3. Sort or tally data in descending order by occurrence or frequency of cause and characteristics.
4. Construct chart – Use scale on “x” and “y” axis to correspond to data collected or sorted as shown in Figure 6-1.

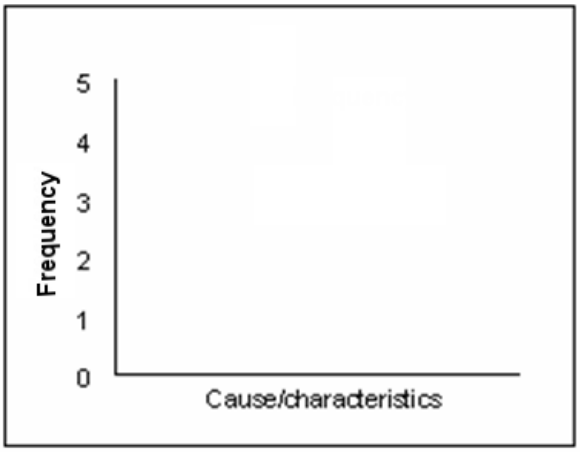


Figure 6-1 Pareto X, Y Chart

5. Draw bars to correspond to sorted data in descending order as shown in Figure 6-2.

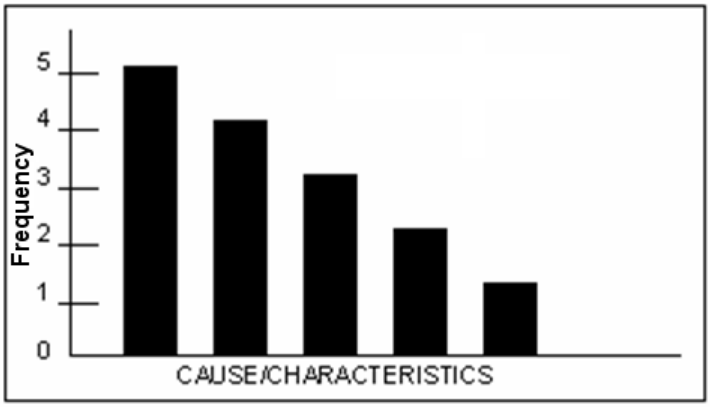


Figure 6-2 Pareto Bar Chart

6. Determine vital few causes (20-80 Rule) as shown in Figure 6-3.

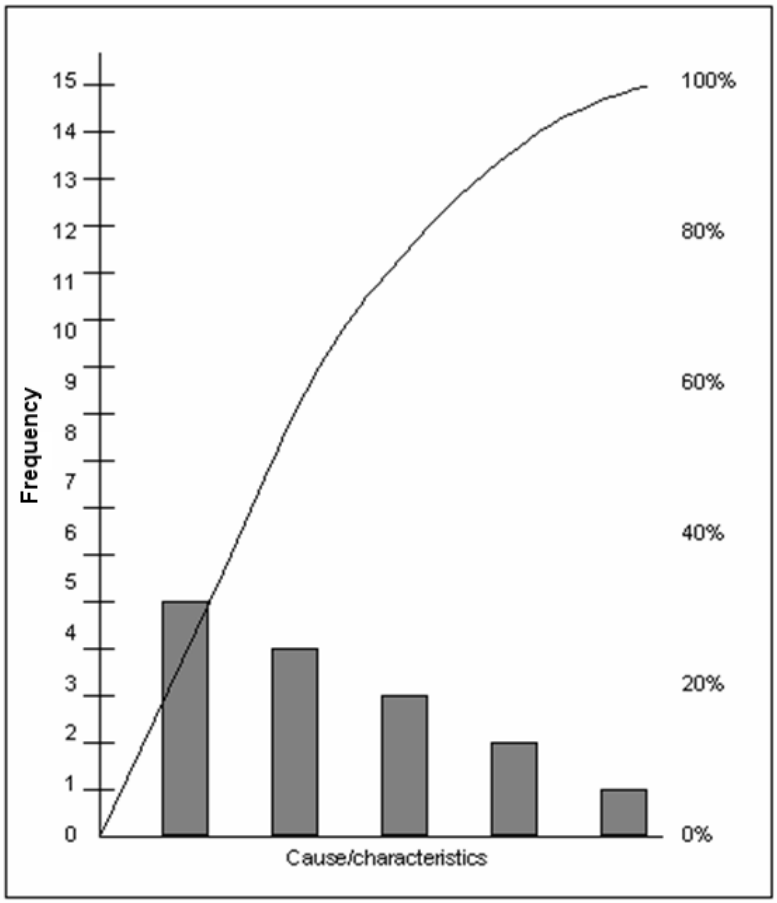


Figure 6-3 Pareto Vital Few Causes Chart

7. Compare and select major causes and repeat process until problem's root causes are reached sufficiently to resolve the problem.

6.2.1.2 Examples

- Problem-solving for vital few causes and characteristics.
- Defect analysis
- Cycle or delivery time reductions
- Unexpected computer processing terminations found in production
- Employee satisfaction or dissatisfaction

6.2.1.3 Results

- A necessary first step in continuous process improvement.

- Graphically demonstrates the 20-80 Rule or vital few number of items that contribute the large percentage (80%) of problems or causes.
- Provides the ability to identify which problem or cause to work on first by its severity or impact.

6.2.1.4 Recommendations

The Pareto chart is a tool that is easy to understand. To use this tool effectively requires discipline by management teams, facilitators, and teams involved with the continuous improvement process.

6.2.2 Pareto Voting

To identify significant potential causes of a problem when the problem cannot be quantified, the Pareto voting technique can be used to obtain the vital few. Use a Pareto chart for quantifiable causes. Like a Pareto chart, Pareto voting is based upon the Pareto Principle – 20% of the potential causes brainstormed will usually be chosen by 80% of the group.

Pareto voting is usually used in conjunction with a cause and effect (fishbone) diagram. This technique is commonly used by management teams that have implemented a quality management process. However, any team can use this tool to separate the vital few causes from the trivial many, as a means of ranking.

6.2.2.1 Deployment

The steps for using this tool are different from Pareto charting, but the same in results – decision-making on when to identify the most potential critical causes of a non-quantifiable problem. The end result is working on the right things first. The following steps are used:

1. Complete brainstorming for potential causes of a problem.
2. Determine the total number of brainstormed ideas and multiply by 20%. For example: 10 ideas results in 2. If number is a fraction, round up to next whole number.
3. Based on the result of Step 2, determine the number of votes each team member receives. In this case, each team member receives two votes.
4. Each team member then uses his or her allocated votes (two in this case) to select the cause(s) having the largest impact on the stated problem.
5. Tally votes each cause receives. Those receiving the most votes are considered most important to the team.
6. Determine the plan of action to resolve these causes.

6.2.2.2 Example

- Determine questions to ask your end user or your employees on satisfaction surveys.
- Determine why quality is or isn't working.
- Determine areas, courses or programs for training.

6.2.3 Cause and Effect Diagrams

Useful tools to visualize, clarify, link, identify, and classify possible causes of a problem is sometimes referred to as a "fishbone diagram," or an "Ishikawa diagram," or a "characteristics diagram." The champion of the use of this diagram was the late Kaoru Ishikawa, a quality leader from Japan.

A team tool used to help identify the causes of problems related to processes, products and services. This technique keeps teams focused on a problem and potential causes. By better understanding problems within the work processes, teams can reach probable and root causes of a problem. A diagnostic approach for complex problems, this technique begins to break down root causes into manageable pieces of a process. A cause and effect diagram visualizes results of brainstorming and affinity grouping through major causes of a significant process problem. Through a series of "why-why" questions on causes, a lowest-level root cause can be discovered by this process.

6.2.3.1 Deployment

Developing a cause and effect diagram requires a series of steps:

1. Generally, as a result of a brainstorm session, identify a problem (effect) with a list of potential causes.
2. Construct a fishbone diagram with basic material: flip-chart, paper, tape, water-base markers, brainstorm cards or post-its.
3. Write the effect (problem) at the right side as shown in Figure 6-4.

Figure 6-4 Effect Problem

4. Identify major causes of the problems, which become “big branches” as shown in Figure 6-5.

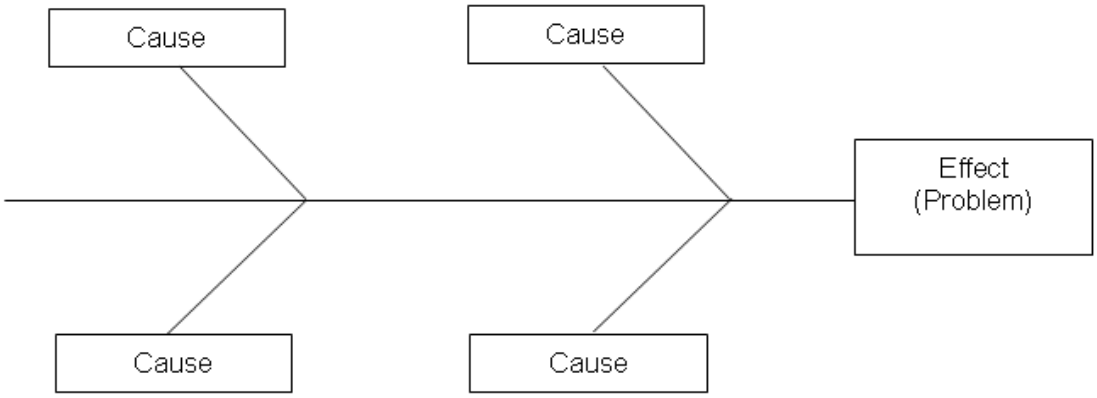


Figure 6-5 Big Branches Problem

5. Use Results of brainstorm or affinity diagram to fill "small branches" as shown in Figure 6-6.

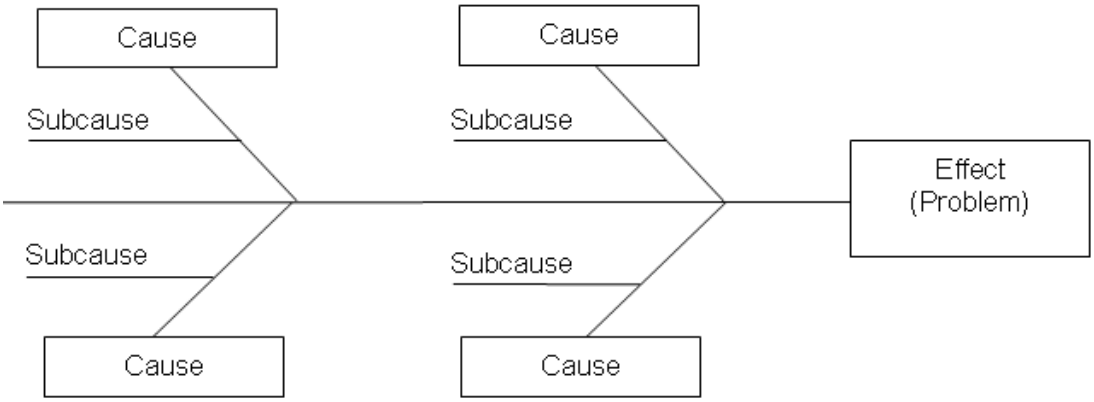


Figure 6-6 Small Branches Problem

6. Complete process until lowest-level sub-cause is identified as shown in Figure 6-7.

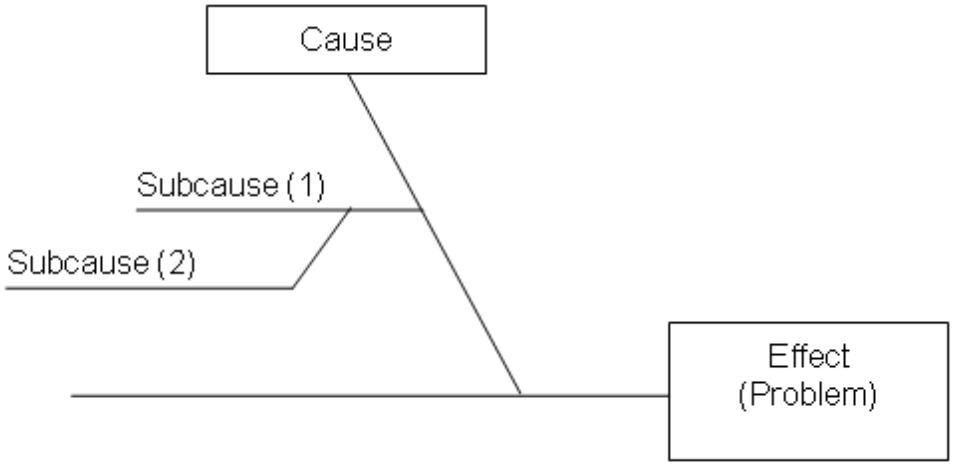


Figure 6-7 Lowest-Level Subcause

7. After the team completes the fishbone diagram, review, check, and verify with the work process that these causes (factors) do strongly affect the problem or causes being resolved.

8. Select most important causes to work on first. Many causes or root causes may need to use nominal grouping or the Pareto voting technique before reaching consensus.
9. Verify root causes by collecting appropriate data (sampling) to validate a relationship to the problem.
10. Continue this process to identify all validated root causes.

6.2.3.2 Results

- Provides a visual relationship between cause and effect.
- Breaks down problem into manageable group of root causes that contribute most to a problem.
- Separates symptoms of a problem from real causes.
- Provides interaction within a team to analyze problems.

6.2.3.3 Examples

- Analysis of problems
- Source for potential process improvements
- Identify sources of defect causes
- Improper use of test routines and testing problems
- Scheduling problems and cycle times
- Compliance to standards and implementation of standards

- Non-manufacturing – Display possible cause of work-related problem or conditions with the 4-Ps: policy, procedures, plant (environment and facilities), and people

6.2.3.4 Recommendation

Use to analyze problem related to workplace or processes owned by a team.

6.2.4 Check Sheets

A check sheet is a technique or tool to record the number of occurrences over a specified interval of time; a data sample to determine the frequency of an event. The recording of data, survey, or sample is to support or validate objectively the significance of the event. This usually follows the Pareto analysis and cause and effect diagram to validate and verify a problem or cause. The team uses this technique in problem solving to support the understanding of a problem, cause, or process. This technique or tool is often used to establish a frequency or histogram chart.

6.2.4.1 Deployment

Developing a check sheet requires a series of steps:

1. Clarify what must be collected objectively.
2. Establish the format for the data collection; one easily understood by the collector.
3. Everyone involved needs to understand the objectives to ensure accuracy of the collection process.
4. Establish the sample size and time frame of data collection.
5. For consistency, instruct or train data collectors.
6. Observe, record, and collect data.
7. Tally results, using Pareto chart or histograms.
8. Evaluate results – a team evaluation process provides better understanding and verification of data collected to support its original analysis.

6.2.4.2 Results

- Provides objective factual data to evaluate problems. Provides causes or processes early in the problem-solving process. Provides a tracking method.
- Detects patterns occurring in the process where problems are suspect. Provides data for Pareto charts or histograms.

6.2.4.3 Examples

- Project review results – defect occurrences, location, and type
- Documentation defects by type and frequency
- Cycle times-requirements to design, design to implementation
- Late deliveries
- End user complaints-all types
- Conformance to standards
- End user surveys

6.2.4.4 Recommendations

Use check sheets as a standard for problem solving whenever data is available or can be collected to validate what is happening to a process or underlining a problem early in the project.

The advantages and disadvantages are listed below:

- Advantages
  - Defines areas to discuss
  - Limits scope
  - Consistency
  - Organized approach
  - Documents results
- Disadvantages
  - Over reliance
  - Applicability
  - Limiting

The following are suggestions for preparing checklists:

- Avoid bias
- Mix questions by topic
- Test questions prior to use
- Allow for "I don't know"

The following are suggestions on using checklists:

- Learn reason for question
- Determine applicability or completeness
- Prepare or rehearse
- Anticipate response
- Ask without questionnaire
- Document when interview complete

6.2.4.5 Example Check Sheet

Figure 6-8 is an example of a check sheet.

Unexpected Computer Processing Terminations	February 21-25					
	Day 1	Day 2	Day 3	Day 4	Day 5	Total

Figure 6-8 Check Sheet Example

6.2.5 Histograms

A histogram is an orderly technique of grouping data by predetermined intervals to show the frequency of the data set. It provides a way to measure and analyze data collected about a process or problem. Pareto charts are a special use of a histogram. When sufficient data on a process is available, a histogram displays the process central point (average), variation (standard deviation, range) and shape of distribution (normal, skewed, and clustered).

Figure 6-9 illustrates a simple histogram.

INTERVAL	TABULATION	FREQUENCY	CUMULATIVE FREQUENCY
0-3	11	2	0-3
3-6	1111 1	6	3-6
6-9	111	3	6-9

Example
Graphic (bar chart)

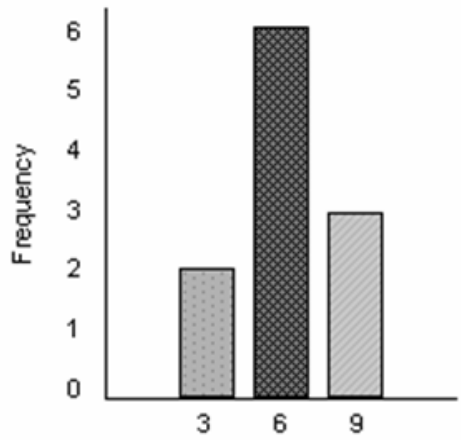


Figure 6-9 Simple Histogram Example

6.2.5.1 Variation of a Histogram

1. Polygon: Draw line from midpoints of bars.
2. Add range of acceptable values (e.g., within plus or minus 5 of budget) to show if actual values lie within acceptable range.

6.2.5.2 Deployment

A histogram requires some understanding of the data set being measured to consolidate and condense into a meaningful display. To do this, the following steps should be taken:

1. Gather data and organize from lowest to highest values.
2. Calculate the range (r): largest less smallest.
3. Determine number of cells (k) – normally between 7 and 13.
4. Calculate the interval or width (m) of the cells: m = range/k.
5. Sort the data or observations into their respective cells.
6. Count the data points of each cell (frequency) to determine the height of the interval.

7. Create a frequency table.
8. Plot the results.
9. Distribution pattern from histograms: normal, double peak, isolated island, cliff, cog-wheel, and skewed.

6.2.5.3 Results

- Helps explain graphically if a process is in or out of control - define a process.
- Provides a basis for what to work on first, especially using the Pareto chart application.
- Provides insight on the process capability to meet end user specifications.
- Establishes a technique to measure a process.
- Analyze for improvement opportunities.

6.2.5.4 Examples

- Defects by type
- Defects by source
- Delivery rates or times
- Experience or skill levels
- Cycle times
- End user survey responses

6.2.5.5 Recommendations

Everyone should use the histogram technique, especially teams that want to understand the nature of the processes they are accountable for or own.

6.2.6 Run Charts

A run chart is a graph of data (observation) in chronological order displaying shifts or trends in the central tendency (average). The data represents measures, counts or percentages of outputs from a process (products or services).

Run charts track changes or trends in a process as well as help to understand the dynamics of a process. This technique is often used before a control chart is developed to monitor a process. A run chart is established for measuring or tracking events or observations in a time or sequence order.

6.2.6.1 Deployment

1. Decide which outputs of a process you need to measure.
2. Label your chart both vertically (quantity) and horizontally (time).
3. Plot the individual measurements over time (once per time interval or as they become available).
4. Connect data points for easy use and interpretation.
5. Track data chronologically in time.
6. Follow these guidelines to interpret certain patterns when monitoring the data points:
    - Unusual events - Eight or more data points above or below the average value indicates the average has changed and must be investigated. If shift is favorable, it should be made a permanent part of the process. If unfavorable, it should be eliminated.

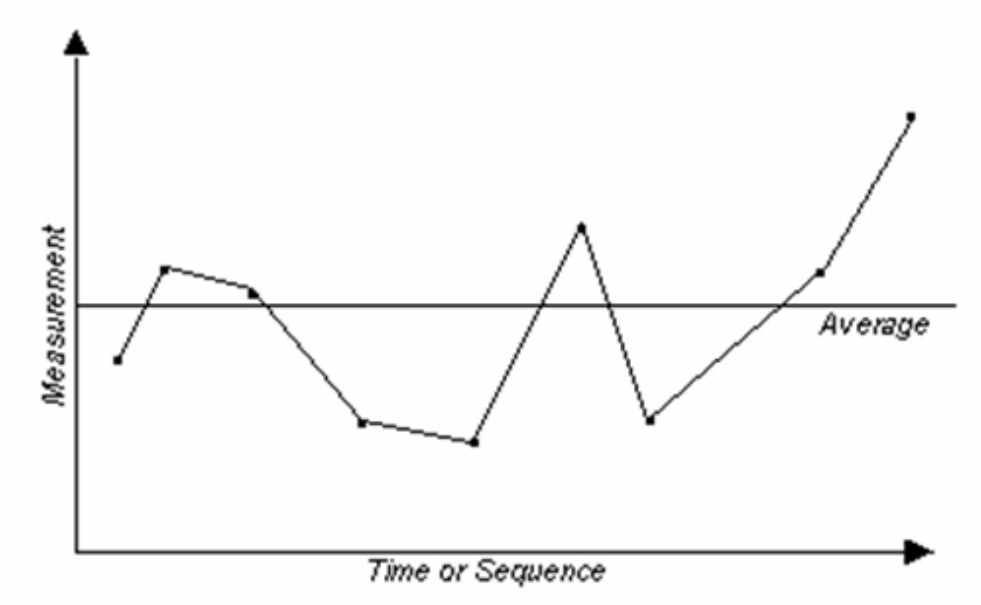


Figure 6-10 Example of a Run Chart

- Trend - Six or more data points of continuous increase or decrease. Neither pattern would be expected to happen based on random chance. Strong indication that an important change has occurred and needs to be investigated.
- Two processes - Fourteen or more data points in a row alternating up or down. Indicates two distinct patterns caused by two groups, two shifts, or two people.
- All special causes need to be investigated to determine these patterns.

6.2.6.2 Results

- Monitor outputs of a process to better understand what is happening from the process.

- Provides a means to detect shift or trends in a process.
- Provides input for establishing control charts after a process has matured or stabilized in time.

6.2.6.3 Examples

- Total unexpected computer processing terminations
- Complaint levels
- End user satisfaction level
- Suggestion levels
- Training efforts
- Production yields
- Number of invoices
- Number of system errors
- Down time (minutes,%)

6.2.6.4 Recommendations

Use to quantify and determine what is happening in or with a process. Several samples of run charts begin the basis for a control chart.

6.2.7 Scatter Plot Diagrams

A scatter plot diagram shows the relationship that might exist between two variables or factors. It can test for possible cause and effect relationships. Walter Shewhart at Bell Laboratories popularized this technique.

This technique explores possible relationships between a variable and a response to test how one variable influences the response. The use of this tool is for problem solving and the understanding of cause and effect relationships. It is often referred to as “correlation” diagrams (positive, negative, or zero). Typical scatter plot diagrams are shown in Figure 6-11.

6.2.7.1 Deployment

This technique is used during analysis to study the relationship between two variables, or cause and effect:

1. Select the variable and response relationship to be examined by the team.
2. Gather data on variable and response; determine sample size of paired data.
3. Plot the results; determine appropriate scale to plot the relationship.
4. Circle repeated data points as many times as they occur.

The pattern of the plots will reveal any correlation such as, positive, negative, random, linear, curvilinear, or cluster. A frequent error in interpreting results is to assume that no relationship exists because a relationship isn't immediately apparent. It may be necessary to take another sample.

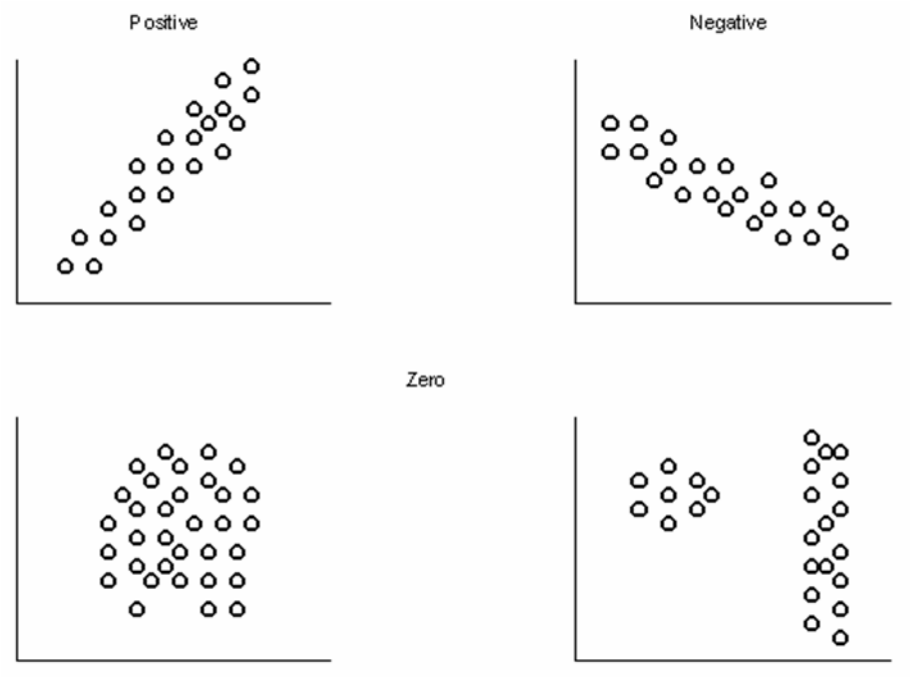


Figure 6-11 Example of a Scatter Plot Diagram

6.2.7.1.1 Using Scatter Diagram for Exploratory Analysis

Computer populations are difficult to conceptualize and to analyze. Manual populations of information or documents are equally difficult. In manual analysis, data is normally randomly sampled, and then the results are extrapolated to draw conclusions about the population. Using computer graphics, random sampling may not be necessary in order to identify unusual conditions.

Exploratory graphics present information from the population of data under review in a manner such that unusual conditions are highlighted.

Exploratory graphics are not meant to replace statistical sampling. They are an adjunct to it, or an alternate means for identifying problems. However, while statistical samples, if performed properly, can be used to predict characteristics of populations, exploratory graphics cannot be used to project the characteristics of the population.

Exploratory graphics are used primarily to identify conditions for additional investigations. While the same result could be achieved by listing and analyzing large amounts of records, the impact of graphics is missing in the listings.

The ability to use scatter diagrams for exploratory analysis is almost unlimited. Three of the more common uses for exploratory analysis are discussed in the following sections.

6.2.7.1.2 Cluster Analysis

Most of the exploratory graphics use scatter plots. A scatter plot shows items, events and transactions on a graph as a point. The scatter concept means that the transactions are scattered in accordance with their characteristics.

The types of scatter charts can be single- or multi-dimensional. In a single-dimensional scatter chart, there is the X and Y-axes with the items scattered within the intersection of those axes. Figure 6-13 shows a scatter plot of this type, while Figure 6-12 expands the scatter plot capabilities by showing the complete X and Y-axes. In Figure 6-12, one axis represents budget, both under and over budget, while the other axis shows schedule, representing behind and ahead of schedule. This scatter plot shows computer project completions.

Each computer project completed is plotted on the chart showing how much over or under the budget is, and how much ahead or behind schedule it is.

The cluster analysis is used to identify groups or clusters of items, which are in close proximity of one another on the scatter plot. For example, the circle in Figure 6-12 shows a cluster of projects over budget and behind schedule. Having identified this cluster, the analyst could now use it to determine if there are unique characteristics among those projects, which cause them to be over budget and behind schedule; for example, they might all use database terminology. Thus, the scatter plot used to identify this cluster could lead the reviewer to a conclusion about database projects.

Cluster analysis is particularly useful in the following instances. See Figure 6-12 for a cluster analysis using a scatter plot.

- Identifying common characteristics of subcategories of events, for example, the over budget and behind schedule of database projects.
- Multi-variable relationships, for example, the budget and schedule relationship.
- Identification of items that should be of concern to management, but they may not have realized the quantity of items in the cluster.

Items to sample because they fall outside expected ranges. Note that many times the cluster is around the intersection of the axis, so those items outside the cluster represent potential problems for investigation.

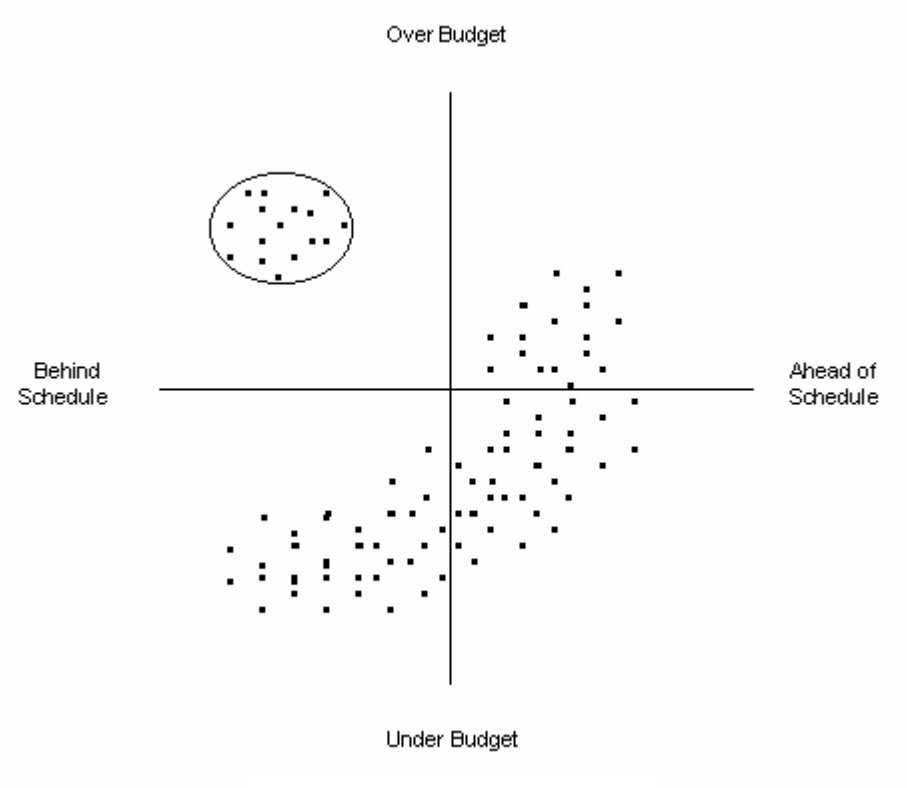


Figure 6-12 Cluster Analysis using a Scatter Plot

6.2.7.2 Results

- Gives analysis between two measurement variables in a process.
- Provides a test of two variables being changed to improve a process or solve a problem.
- Helps to recover real causes, not symptoms, of a problem or process.

6.2.7.3 Examples

- Defect level versus complexity.
- Defects versus skill levels (training).
- Failures versus time.
- Cost versus time.
- Change response versus people availability.
- Defect cost versus where found (life cycle).
- Preventive cost versus failure cost.

6.2.8 Regression Analysis

Regression analysis is a means of showing the relationship between two variables. Regression analysis will provide two pieces of information. The first is a graphic showing the relationship between two variables. Second, it will show the correlation, or how closely related the two variables are.

The regression analysis shown in Figure 6-13 illustrates the relationship between size of computer programs as expressed in number of lines of executable code and the cost to develop those programs in dollars. In this illustration, there is almost a perfect correlation between the size of a computer program and the cost to produce that program. The correlation shows that as the size of the program increases, the cost to develop that program increases. While the information presented is not related to specific size in dollars, the conclusion drawn has been statistically validated through regression analysis.

Regression analysis is used in analysis to show cause and effect relationships between two variables. In many instances, the analysis may hypothesize that such a relationship is correct, but cannot support it without regression analysis. The use of graphics that show these correlations are not only valuable for analysis, but are valuable for presentation. For example, if the analyst wanted to recommend that there be a standard or guideline on size of computer programs, this graphic would illustrate to management the cost of not following that guideline.

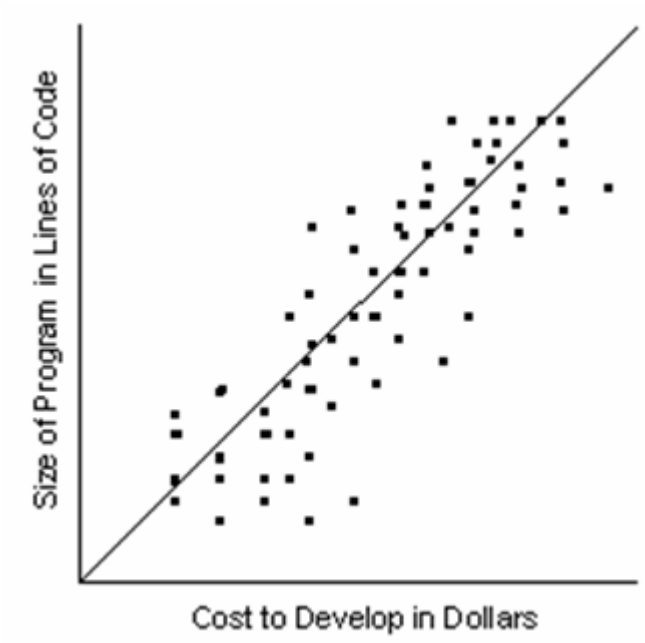


Figure 6-13 Regression Analysis using a Scatter Plot

6.2.8.1 Deployment

Regression analysis requires the use of a statistical software package, coupled with graphic presentation. The math required for regression analysis is usually substantial.

Note that our regression analysis example is a regression of two variables. In actual practice, the analyst may wish to pick three or more variables in order to show regression. When more variables are used, it is important to calculate the standard error that is attributable to using the regression formula.

The standard error deals with the probability that the regression is correct. Generally, the smaller the standard error, the more closely the variables are related. However, the graphic also helps show the standard error. The more closely related the scatter plot to the regression line, the less the standard error shown by that line.

6.2.8.2 Results

Among the uses for regression analysis is:

- Cause and effect relationship of two variables, for example, the effect on cause of the size of a computer program.
- Projecting performance based on multiple variables. For example, predicting what an individual's salary should be, based on the background, skill, and other performance characteristics of that individual.
- Probing to determine the cause of some unfavorable event, such as computer projects significantly missing schedules. By developing correlations among multiple variables, the analyst can begin to see which variable may be causing the undesired result.

6.2.9 Multivariate Analysis

A multivariate analysis shows the relationship between two variables. The multivariate analysis example illustrated in Figure 6-14, shows two variables presented on the same chart. One line shows the dollar sales per salesperson, and the other line, the profit on sales. The conclusion from this graphic would be that the more volume a salesperson sells, the less profit made. Obviously, if this conclusion is correct some underlying information is needed. For example, the reason may be that there is less profit in the high-dollar area.

Reading these graphics begins by examining the shape of distribution of single variables. In Figure 6-14, we would need to look at the distribution of profits and the distribution of sales among sales personnel. Remember that while our example appears to show a meaningful relationship between these multiple variables, each graphic will not necessarily produce meaningful information.

The concept of “exploratory graphics” is used because many times the graphical information produced is not helpful in identifying areas for further analysis. Figure 6-14 has distributions that are distributed in a manner that makes analysis meaningful. On the other hand, different distributions would show random lines, meaning that some salespeople with high-dollar volumes had high profits, while other salespeople with high-dollar volumes had low profits. The conclusion from such a graphic would be meaningless.

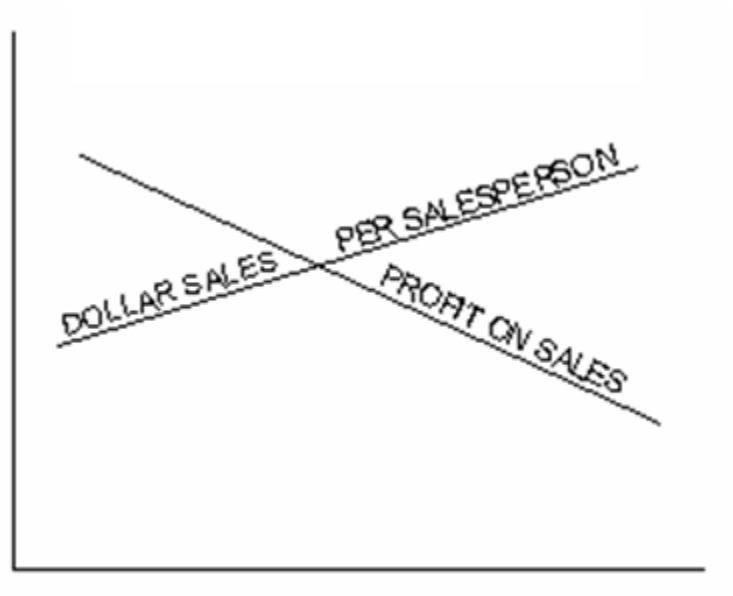


Figure 6-14 Multivariate Analysis

6.2.9.1 Deployment

The multivariate analysis normally requires advanced statistical packages to perform the underlying analysis. In most instances, regression statistics are used to produce the information needed for creating the multi-variable graph.

Exploratory graphics doe not confirm hypotheses but, rather, poses hypotheses. The results of exploratory graphics must be confirmed through more investigation. Only after this confirmation should these exploratory graphics be used in presentations and reports.

6.2.9.2 Results

Multivariate analysis can be used:

- Only after individual variables have been carefully examined.
- When scatter plots of the individuals have been made and the information smoothed in order to examine or plot the individual variables.

6.2.10 Control Charts

A statistical technique to assess, monitor and maintain the stability of a process. The objective is to monitor a continuous repeatable process and the process variation from specifications. The intent of a control chart is to monitor the variation of a statistically stable process where activities are repetitive. Two types of variation are being observed: 1) common, or random; and, 2) special or unique events.

Control charts are used to evaluate variation of a process to determine what improvements are needed. Control charts are meant to be used on a continuous basis to monitor processes.

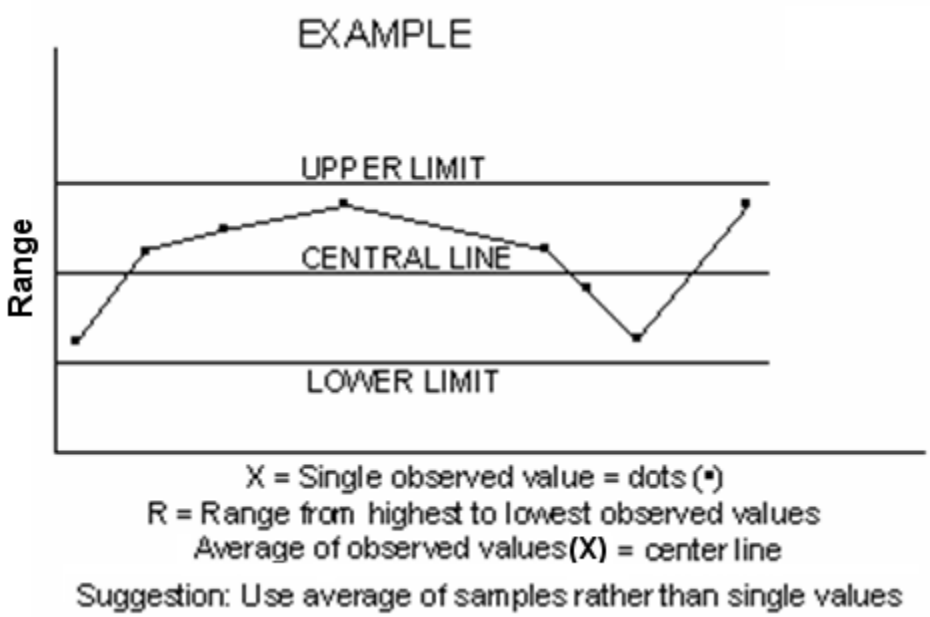


Figure 6-15 Control Chart

6.2.10.1 Deployment

A decision to use control charts is serious business and should not be taken lightly. Normally, this occurs when a process is thought to be out of control. Initially, a team evaluates what is going wrong by using brainstorming, Pareto analysis, and cause and effect diagrams to understand the problem. The next steps lead into control chart application:

1. Identify characteristics of process to monitor: defects, cycle times, unexpected computer processing terminations, cost, or maintenance.
2. Select the appropriate type of control chart based on characteristics to monitor.
3. Determine methods for sampling – how many, over what time frame; use check sheets.
4. Collect sample data.
5. Analyze and calculate sample statistics: average, standard deviation, upper limit, lower limit.
6. Construct control chart based on statistics.
7. Monitor process for common and special causes.
8. Since a process is in control when observations fall within limits, evaluate and analyze any observation outside the limits for causes related to the situation.
9. Investigate unusual patterns when observations have multiple runs above or below central line (average). A process shift is occurring and the reason needs to be understood. This may lead to process improvements or corrections.

6.2.10.2 Results

- Objectively defines a process and variation.
- Establishes measures on a process.
- Improves process analysis and opportunities.
- Process improvements are based on facts-managed by facts.

6.2.10.3 Examples

- Unexpected computer processing terminations in production.
- Defects by life cycle phase.
- Complaint or failures by application or software.
- Response time to change request.
- Cycle times or delivery times.
- Mean time to failure.

6.3 Test Tools used to Enhance Test Reporting

Project managers, IT management and users of software often judge the effectiveness of software testing by the reports issued by testers. The more useful information that testers can convey in their test reports, the more favorable impression of testing that is achieved. There are many tools available to help in report writing such as report graphics, color, highlighting key points and management summaries.

Two tools that are not normally used by testers, but which are helpful in report writing, are described in this category:

- The first is benchmarking to illustrate a comparison to other projects tested internally, as well as testing results and approaches by external test organizations.
- The second is Quality Function Deployment (QFD) that allows testers to trace requirements throughout development.

6.3.1 Benchmarking

Benchmarking is the continuous process of measuring products, services, and practices against our toughest competitors, or those companies recognized as world leaders. It is a way to achieve process improvement, motivation, and a management process for improvement.

Benchmarking is not normally associated with cost cutting, stealing or copying a difficult activity, or a quick fix. The purpose of benchmarking is to:

- Gain a better awareness of yourself:

  - What you are doing and how
  - How well you are doing it
- Gain a better awareness of "the best":
  - What they are doing and how
  - How well they are doing it
  - Identify the performance gap
- Understand how to change business processes to improve quality, cost, and delivery.

The three types of benchmarking are explained below. Note that process and product benchmarking account for approximately 80 percent of all benchmarking.

- Performance Benchmarking

  Use performance benchmarking to set and validate objectives for key performance metrics and for projections of improvements required to close the benchmark "gap."

- Process Benchmarking

  Use Process benchmarking to plan for business process improvement and document as part of business plans and quality improvement projects.

- Product Benchmarking

  Use Product benchmarking to help in product planning and development. Product documentation includes the product performance goals and design practices identified through benchmarking.

6.3.1.1 A Ten-Step Process to Collect Benchmark Data

Benchmarking is a ten-step process, involving four phases, as described in Figure 6-16.

6.3.1.1.1 Planning Phase

1. *Identify benchmarking subject and teams*. These can be internal or external candidates. The candidates come from personal knowledge, interaction with industry groups, studying industry reports; and interviewing consultants, professional groups, and so forth.
2. *Identify and select benchmarking partners*. The purpose of this step is to determine viable candidates for benchmarking; gain participation agreement by phone calls; and confirm visit time period, agenda, and attendees with the benchmarking partner.

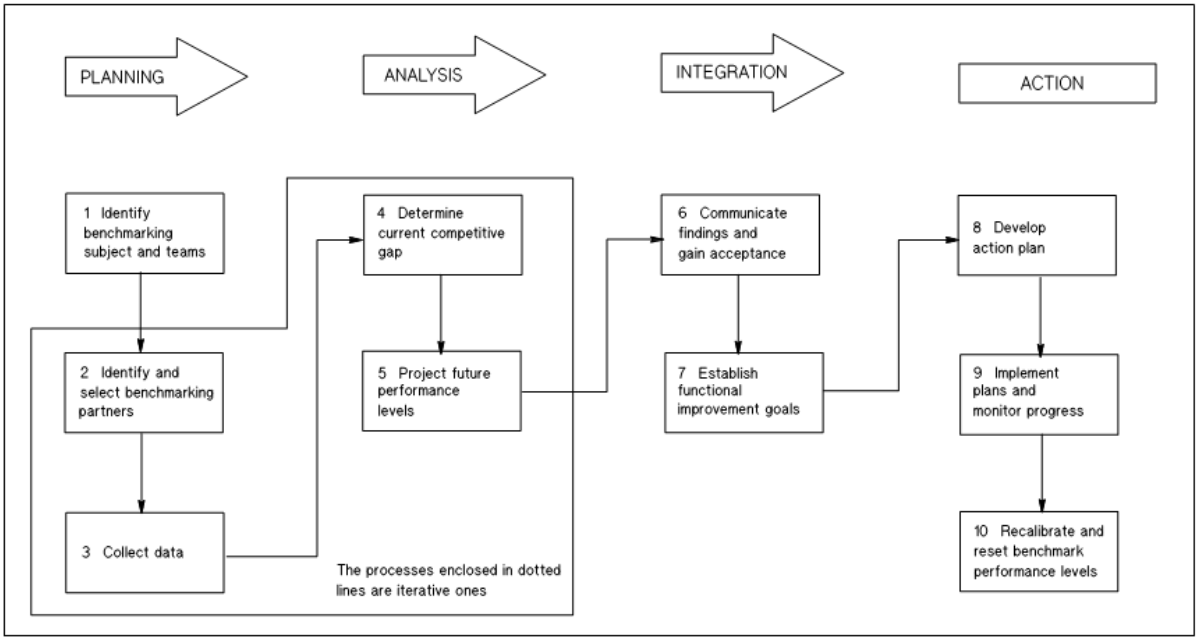


Figure 6-16 10-Step Benchmarking Process

3. *Collect data.* The performance of this step requires you to document your own process, develop question lists, meet with the process owners to collect data, and record the data.

6.3.1.1.2 Analysis Phase

4. *Determine current competitive gap*. Determine the difference between the attributes of your own process, product, performance, and that of the benchmark partner. This is done for many different attributes of a process, product, or performance.
5. *Project the future performance levels*. Make a managerial decision of the goals your organization wishes to establish for improved performance based on the competitive gap.

6.3.1.1.3 Integration Phase

6. *Communicate findings and gain acceptance.* Describe the results of benchmarking to the process owners and involved parties, as well as communicating the potential future performance levels to gain acceptance to move forward.
7. *Establish functional improvement goals*. In conjunction with the process owners, establish specific improvement goals to be achieved. (These generally should be short-term goals, not to exceed one year.)

6.3.1.1.4 Action Phase

8. *Develop an Action Plan.* Use your organization's process improvement process to plan improvement.

9. *Implement plans and monitor progress.* Perform the plan, measure progress, and make adjustments as necessary.
10. *Re-calibrate and reset benchmark performance levels*. Based on the analysis, set new goals, re-benchmark to find better ways to do the process, set new goals, and continue the improvement cycle.

There are three ways benchmarks can be deployed:

1. Competitive Benchmarks

   Compare your business performance within the IT industry.
2. Functional Benchmarks

   Compare your business performance with that of the "best in class" or "best in breed" within any industry.
3. Internal Benchmarks

   Compare business performance with that of other company units.

There are three results that normally occur when you benchmark:

1. An assessment of how good you are against industry leaders.
2. An indication of an improvement goal that is realistic based on industry practices. Note that many organizations set goals above the industry benchmark.
3. Insight into how to achieve improvement from analysis of benchmarking partners' processes.

Some examples where benchmarking has been used effectively in IT are:

- Benchmarking to evaluate and upgrade the end-user requirements process.
- Benchmarking to design a professional career ladder for information professionals.
- Benchmarking to identify and install metrics to measure quality and productivity.

There are four lessons to learn from benchmarking from the benchmarking leaders:

1. It’s important to focus on a specific objective and process:
   - Breadth for context; depth for understanding
   - Facilitation of the benchmarking session to keep on track
2. It's key to prepare in advance:
   - Objectives, agenda, date, attendees
   - Meeting and benchmarking protocol
   - Process documentation (own company and partners)
3. It's not easy to identify the IT "best of the best":
   - Good performance data is not readily available
   - Research is required to evaluate opinion versus fact
4. It always takes longer than you think!

6.3.2 Quality Function Deployment

Quality function deployment (QFD) is the only comprehensive quality system aimed specifically at satisfying the end user. It concentrates on maximizing end-user satisfaction (positive quality) – measured by metrics, such as end-user compliments. QFD focuses on delivering value by understanding the end user's wants and needs, and then deploying these expectations downstream in a visible way. With QFD, we design value into the software.

An organized approach to quality with tools, techniques and a set of methods is a quality system. Dr. Yoji Akao, the principal developer of QFD, defines QFD as a quality system with many components as illustrated in Figure 6-17. QFD includes tools and techniques for improving the software product. Comprehensive quality deployment includes quality deployment, technology deployment, cost/schedule deployment, and reliability deployment.

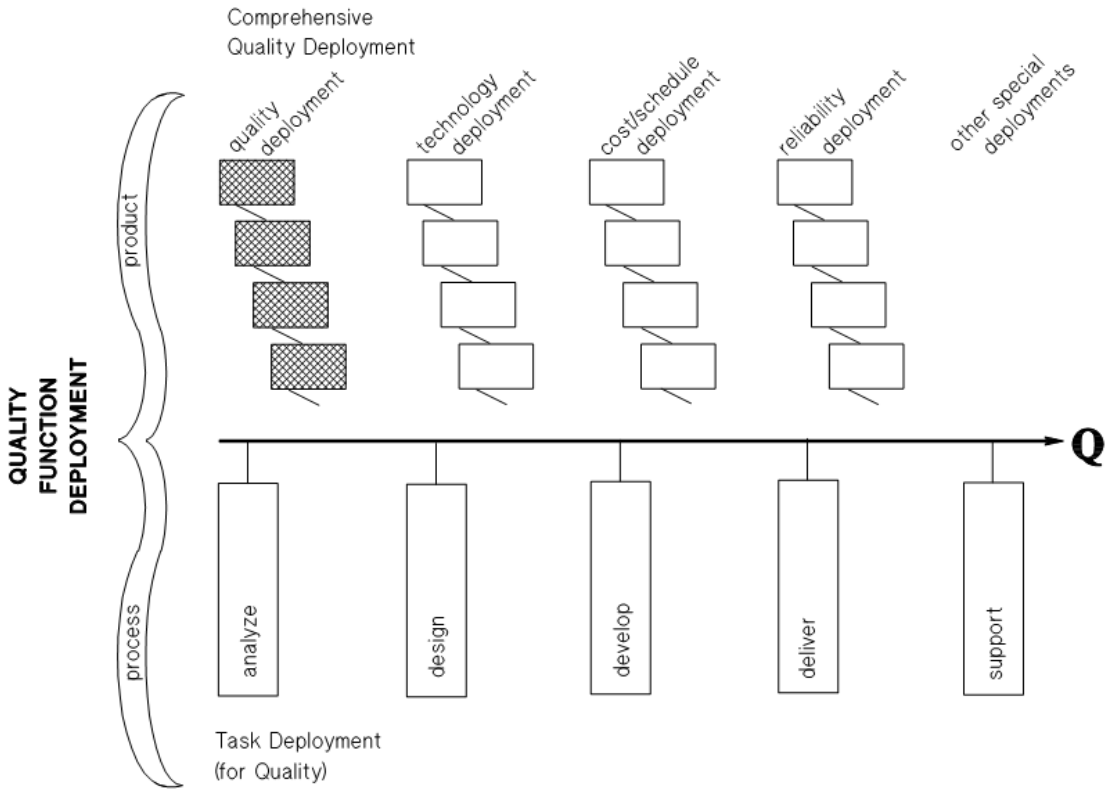


Figure 6-17 Quality Function Deployment Defined

QFD improves both the software product and the software process to better satisfy customers and stakeholders.

It can also address other special concerns with a corresponding deployment – such as usability, reuse, and security. QFD provides forward and backward traceability of value in the software development life cycle. Value is why end users want software.

A special challenge exists for complex products and services – those involving a combination of hardware, software, and service, or those where nontrivial design decisions must be made at the system, subsystem, and component levels. In QFD, there are series of matrices that comprise a visible means to address a particular concern, such as reliability, during development. These deployments are a structured way of dealing with a special concern at a detailed level. They provide a basis for linking the concerns into an integrated framework. This framework results in a very sophisticated (and efficient) product development process.

The starting point in a new product development is the decision of what is to be built for whom. This requires deciding whom the end user is, finding out what they want, and determining what capabilities we can provide them. In QFD, the fundamental deployments of end users and quality address this.

The end-user deployment involves the determination of which types of end users the organization is trying to provide a product/service for. It precedes the determination of requirements in quality deployment. You must first decide who your end users are so you know what voices to listen to. The end-user deployment component of QFD is particularly important for software, as a single software product must often satisfy many types of end users.

The quality deployment has tools and techniques for the exploration and specification of high-value end-user requirements (or "demanded quality"). Once captured, the end-user requirements are translated and deployed into technical requirements (or "quality characteristics") in the A-1 or "house of quality" matrix. This can be done at various levels of sophistication, ranging from four matrices to a dozen.

These fundamental deployments are the foundation for downstream design-and-development decisions about "how" the product will work – the horizontal deployments – and "how well" it will be designed and developed – the vertical deployments.

Dr. Deming said, “We must view development as a system, and look at how we satisfy our end users as an organization.” Software QFD is one quality system with precisely that aim – to deliver great software-intensive products and services to multiple types of end users. A number of leading software firms in Europe, Japan, and North America has applied this approach. Results to date are very promising, and further refinement is still occurring.

The description of QFD was extracted from "Quality Function Deployment (QFD) for Software" by Richard E. Zultner, American Programmer, February 1992, pp. 1-12; and was reprinted with permission of the author.

6.4 Reporting Test Results

Reporting test results should be a continuous process. Whenever significant problems are encountered they should be reported to the decision-makers who can determine the

appropriate action. Testing reports should also be prepared at pre-defined checkpoints and at the end of testing.

In preparing test reports testers should answer these questions:

- What information do the stakeholders need?
- How can testers present that information in an easy-to-understand format?
- How can I present the information so that it is believable?
- What can I tell the stakeholder that would help in determining what action to take?

The following aspects of test reporting are covered in this section:

- Current status test report
- Final test reports

The test reports indicating the current status of reporting, or interim test reports are needed for project management. Those responsible for making project decisions need to know the status from the tester's perspective throughout the project. These interim reports can occur in any phase of the life cycle, at pre-defined checkpoints, or when important information needs to be conveyed to developers.

The final test reports are prepared at the conclusion of each level of testing. The ones occurring at the end of unit and integration testing are normally informal and have the primary purpose of indicating that there are no remaining defects at the end of those test levels. The test reports at the conclusion of system testing and acceptance testing are primarily for the customer or user to make decisions regarding whether or not to place the software in operation. If it is placed in operation with known defects the user can develop strategies to address potential weaknesses.

6.4.1 Current Status Test Reports

Testers need to develop reports that show the status of testing throughout the process of testing the software. The test process should produce a continuous series of reports that describe the status of testing. The current status test reports are for use by the testers, the test manager, and the software development team. The frequency of the test reports should be at the discretion of the team, and based on the extensiveness of the test process. Generally, large projects will require more interim reporting than will small test projects with a very limited test staff.

Thirteen current status reports are proposed here. Testers can use all thirteen or select specific ones to meet individual test needs. However, it is recommended that, if available, test data permits at the end of the testing phase, all thirteen test reports be prepared and incorporated into the final test report. Each of the thirteen reports is described in the following pages, with examples.

6.4.1.1 Function Test Matrix

The function test matrix is the core of a test report. The function test matrix shows which tests must be performed to validate the functions. Its matrix will be used to determine which tests are needed, as well as their sequencing. It will also be used to determine the status of testing.

Many organizations use a spreadsheet package to maintain test results. The intersection can be color coded or coded with a number or symbol to indicate the following:

1. Test is needed, but not performed
2. Test is currently being performed
3. Test was performed and a minor defect noted
4. Test was performed and a major defect noted
5. Test complete and function is defect-free for the criteria included in this test

6.4.1.1.1 Report Example

The function test matrix can be prepared with one of the five test results in the appropriate intersection. The matrix intersections can be color coded to show the results of testing using a color for the five types of test results. The matrix example shown in Table 6-1, “Function Test Matrix Report Example” uses checkmarks to indicate that a test is to be conducted on the functions indicated. The check mark can then be supplemented or replaced to show the results of testing.

Table 6-1: Function Test Matrix Report Example

	Test									
Test Function	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A	√			√				√		√
B			√		√				√	
C		√				√	√			
D		√							√	
E	√		√				√			√

6.4.1.1.2 How to Interpret the Report

The report is designed to show the results of performing a specific test on a function. A low-level report indicates the results of each test. The report is designed to show the status of each test; therefore no interpretation can be made about the results of the entire software system, only about the results of individual tests. However, if all of the tests for a specific function are successful, one could assume that function works. Nevertheless, “working” means that it has met the criteria in the Test Plan.

6.4.1.2 Defect Status Report

A report is needed for each defect found by testers. It is recommended that the information collected about each defect be maintained electronically so that test managers and users of the software system can review it. The information collected about each defect can be as simple or as complex as desired. For simplification purposes, it is suggested that the following guidelines be used:

- Defect Naming

  Name defects according to the phase in which the defect most likely occurred, such as a requirements defect, design defect, documentation defect, and so forth.

- Defect Severity

  Use three categories of severity as follows:

  - Critical – Would stop the software system from operating
  - Major – Would cause incorrect output to be produced
  - Minor – Would be a problem, but would not cause improper output to be produced, such as a system documentation error

- Defect Type

  Use the following three categories:

1. Missing – A specification not included in the software
2. Wrong – A specification improperly implemented in the software
3. Extra – Element in the software not requested by a specification

6.4.1.2.1 Report Example

The defect status report information should be completed each time the testers uncover a defect. An example of a defect status report is shown in Table 6-2, “Defect Status Report”.

Table 6-2: Defect Status Report

Software/System Tested	Name of software being tested.
Date	Date on which the test occurred.
Defect Found (Name/ Type)	The name and type of a single defect in the software being tested.
Location Found (Unit/Module)	The individual unit or system module in which the defect was found.
Severity of Defect	Critical, major, or minor. Critical means the system cannot run without correction; major means the defect will impact the accuracy of operation; minor means it will not impact the operation.
Type of Defect	Whether the defect represents something missing, something wrong, or something extra.
Test Data/Script Locating Defect	Which test was used to uncover the defect.
Origin of Defect/ Phase of Development	The phase in which the defect occurred.
Date Corrected	The date on which the defect was corrected.
Retest Date	The date on which the testers were scheduled to validate whether the defect had been corrected.
Result of Retest	Whether the software system functions correctly and the defect no longer exists; or if additional correction and testing will be required. If so, the “To be added later” section will need to be reentered.

6.4.1.2.2 How to Interpret the Report

The report is designed to both describe a defect and to report the current status of that defect. The individual responsible for the function containing the defect needs to decide what action(s) to take, make any desired correction, and retest the function if changed.

The information from the defect status report can be used to produce the function/test matrix as shown in Table 6-1, “Function Test Matrix Report Example”. In this example, the intersection between the function and test is check marked to indicate test status. Status can be color-coded or numbered 1 to 5, as described earlier.

6.4.1.3 Functional Testing Status Report

The purpose of this report is to list the percent of the functions that have been fully tested; the functions that have been tested, but contain errors; and the functions that have not been tested. The report should include 100 percent of the functions to be tested in accordance with the Test Plan.

6.4.1.3.1 Report Example

A sample of this Test Report is illustrated in Figure 6-18. It shows that 50 percent of the functions tested have errors, 40 percent were fully tested, and 10 percent were not tested.

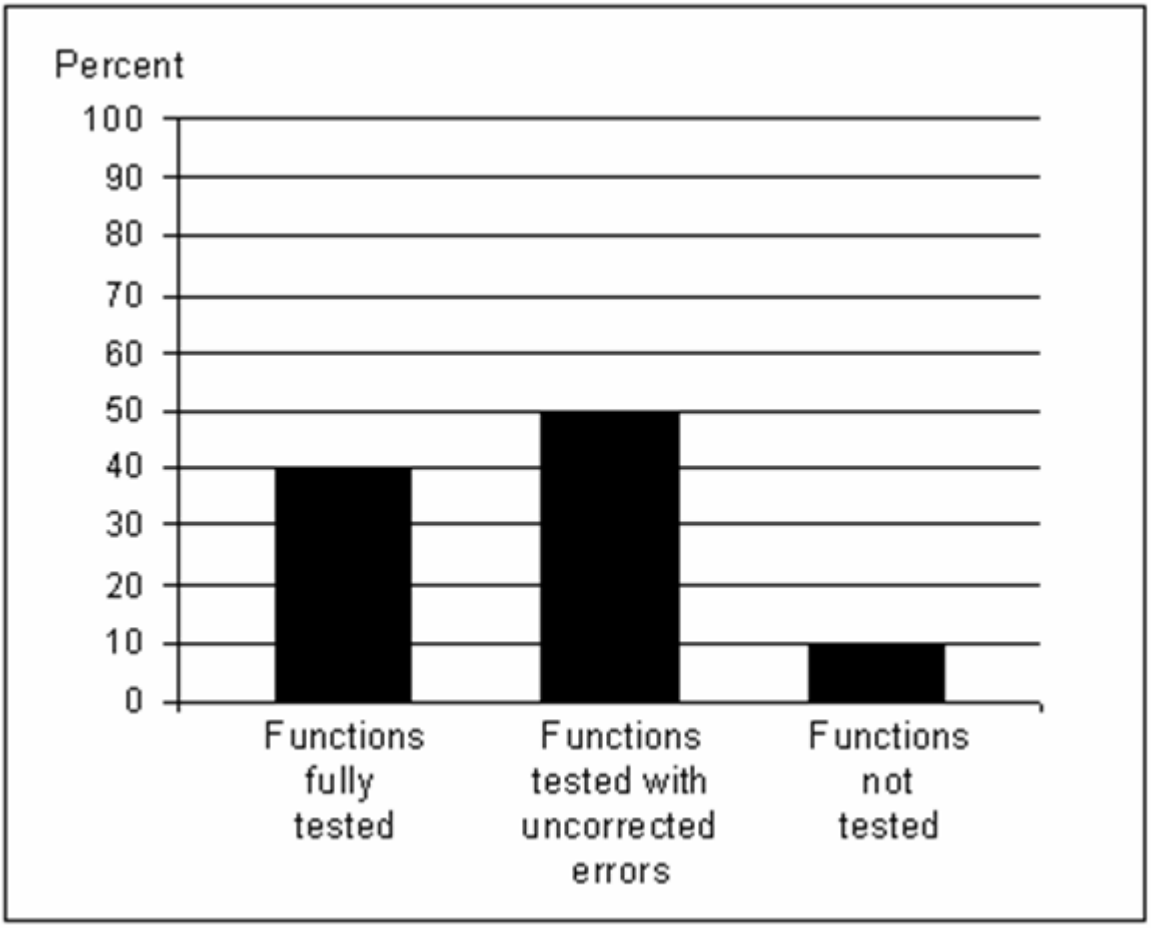


Figure 6-18 Functional Testing Status Report

6.4.1.3.2 How to Interpret the Report

The report is designed to show status to the test manager and customer of the software system. How status is interpreted will depend heavily on the point in the test process at which the report was prepared. As the implementation date approaches, a high number of functions tested with uncorrected errors and functions not tested should raise concerns about meeting the implementation date.

6.4.1.4 Functions Working Timeline

The purpose of this report is to show the status of testing and the probability that the development and test groups will have the system ready on the projected implementation date.

6.4.1.4.1 Report Example

The example of the Functions Working Timeline in Figure 6-19 shows the normal projection for having functions working. This report assumes a September implementation date and shows from January through September the percent of functions that should be working correctly at any point in time. The actual line shows that the project is doing better than anticipated.

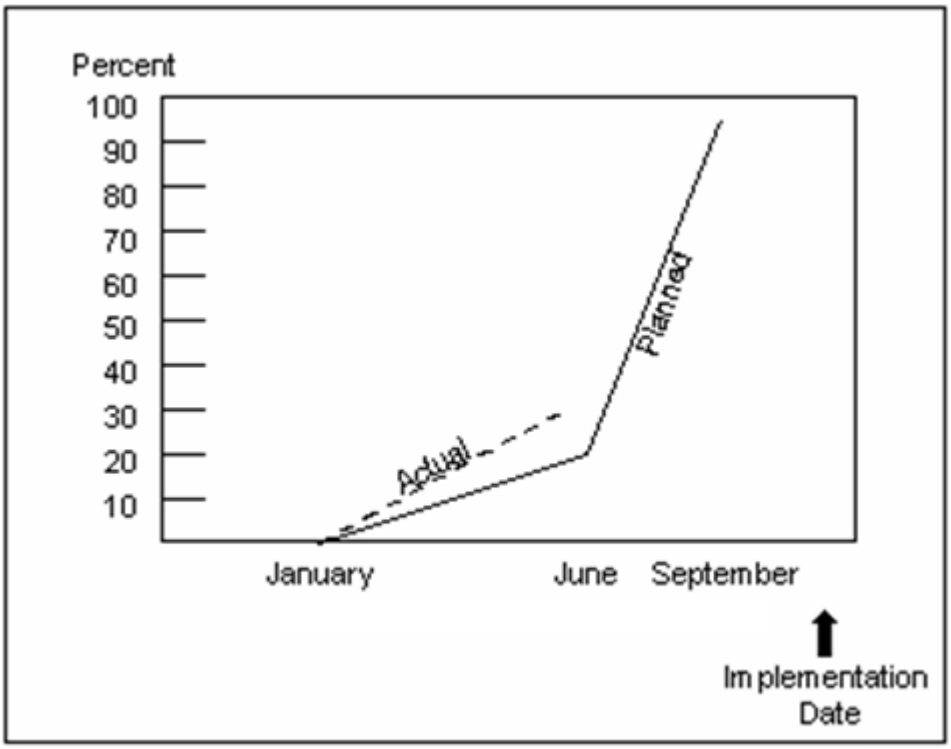


Figure 6-19 Functions Working Timeline Report

6.4.1.4.2 How to Interpret the Report

If the actual performance is better than planned, the probability of meeting the implementation date is high. On the other hand, if the actual percent of functions working is less than planned, both the test manager and development team should be concerned, and may want to extend the implementation date or add resources to testing and/or development.

6.4.1.5 Expected versus Actual Defects Uncovered Timeline

The purpose of this report is to show whether the number of defects uncovered is above or below the expected number. This assumes that the organization has sufficient historical data to project defect rates. It also assumes that the development process is sufficiently stable so that the defect rates from that process are relatively consistent.

6.4.1.5.1 Report Example

The example chart for the Expected versus Actual Defects Uncovered Timeline in Figure 6-20 shows a project beginning in January with a September implementation date. For this project, 500 defects are expected; the expected line shows the cumulative anticipated rate for uncovering those defects. The actual line shows that a higher number of defects than expected have been uncovered early in the project.

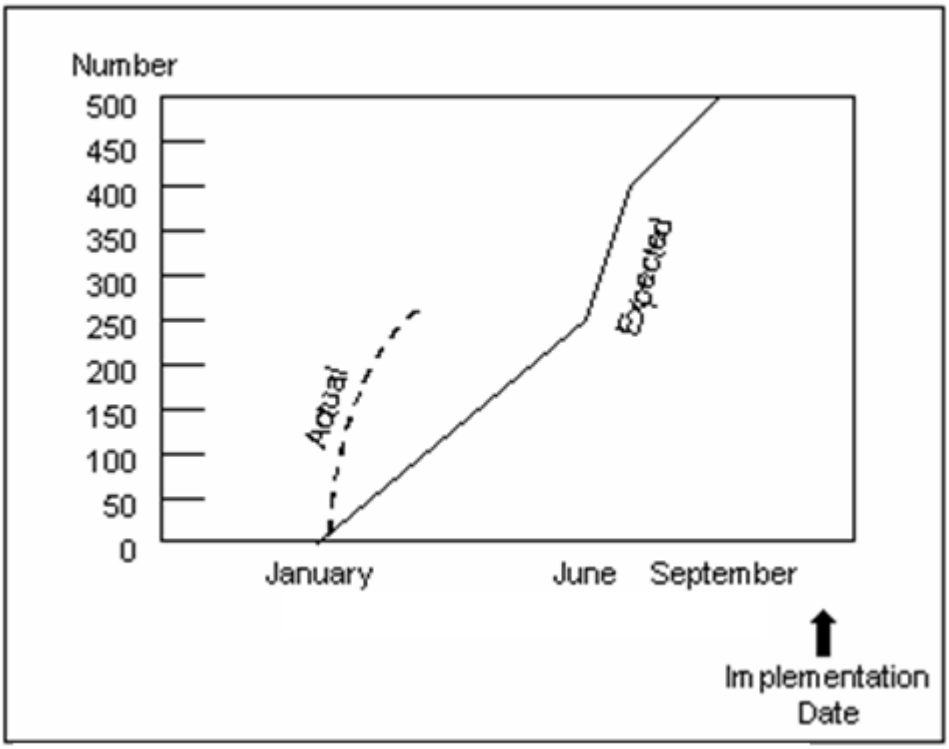


Figure 6-20 Expected versus Actual Defects Uncovered Timeline Report

6.4.1.5.2 How to Interpret the Report

If the actual defect rate varies from the expected rate, generally, there is a special cause, and investigation is warranted. In Figure 6-20, the cause may be because a very inexperienced project team is developing the software. Even when the actual defects are significantly less than expected, testers should be concerned, because it may mean that the tests have not been effective and therefore a large number of undetected defects remain in the software.

6.4.1.6 Defects Uncovered versus Corrected Gap Timeline

The purpose of this report is to list the backlog of detected but uncorrected defects. It requires recording defects as they are detected, and then again when they have been successfully corrected.

6.4.1.6.1 Report Example

The example in Figure 6-21 shows a project beginning in January with a projected September implementation date. One line on the chart shows the cumulative number of defects uncovered during testing, and the second line shows the cumulative number of defects corrected by the development team, which have been retested to demonstrate that correctness. The gap represents the number of uncovered but uncorrected defects at any point in time.

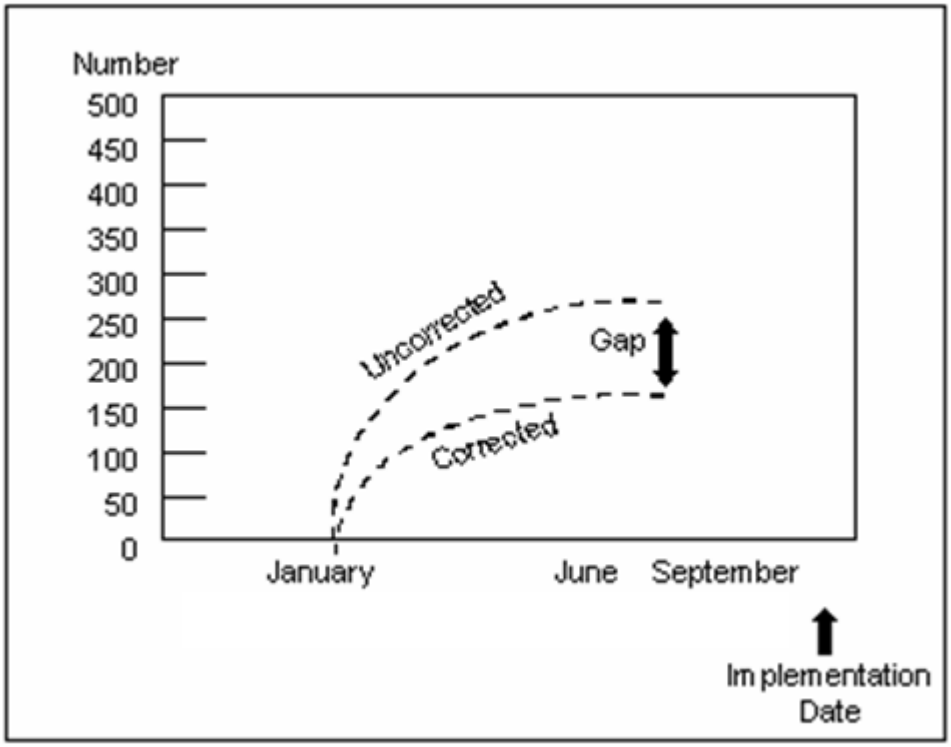


Figure 6-21 Defects Uncovered versus Corrected Gap Timeline Report

6.4.1.6.2 How to Interpret the Report

The ideal project would have a very small gap between these two timelines. If the gap becomes wide, it indicates that the backlog of uncorrected defects is growing, and that the probability the development team will be able to correct them prior to implementation date is decreasing. The development team must manage this gap to ensure that it remains narrow.

6.4.1.7 Average Age of Uncorrected Defects by Type

The purpose of this report is to show the breakdown of the gap presented in Figure 6-21 by the number of days it has taken to correct defects.

6.4.1.7.1 Report Example

The Average Age of Uncorrected Defects by Type report example in Figure 6-22 shows the three severity categories aged according to the average number of days since the defect was detected. For example, it shows that the average critical defect is about 3 days old, the average major defect is about 10 days old, and the average minor defect is about 20 days old. The calculation is to accumulate the total number of days each defect has been waiting to be corrected, divided by the number of defects. Average days should be working days.

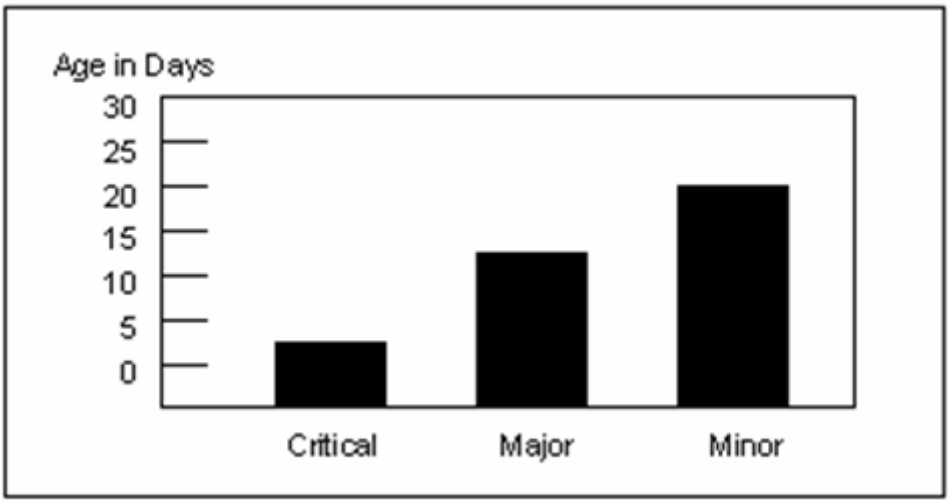


Figure 6-22 Average Age of Uncorrected Defects by Type Report

6.4.1.7.2 How to Interpret the Report

Figure 6-22 is the desirable status, demonstrating that critical defects are being corrected faster than major defects, which are being corrected faster than minor defects. Organizations should have guidelines for how long defects at each level should be maintained before being corrected. Action should be taken accordingly based on actual age.

6.4.1.8 Defect Distribution Report

The purpose of this report is to explain how defects are distributed among the modules/units being tested. It lists the total cumulative defects uncovered for each module being tested at any point in time.

6.4.1.8.1 Report Example

The Defect Distribution Report example in Figure 6-23 shows eight units under test along with the number of defects uncovered in each of those units to date. The report could be enhanced to show the extent of testing that has occurred on the modules, for example, by color-coding the number of tests; or by incorporating the number of tests into the bar as a number, such as 6 for a unit that has undergone six tests when the report was prepared.

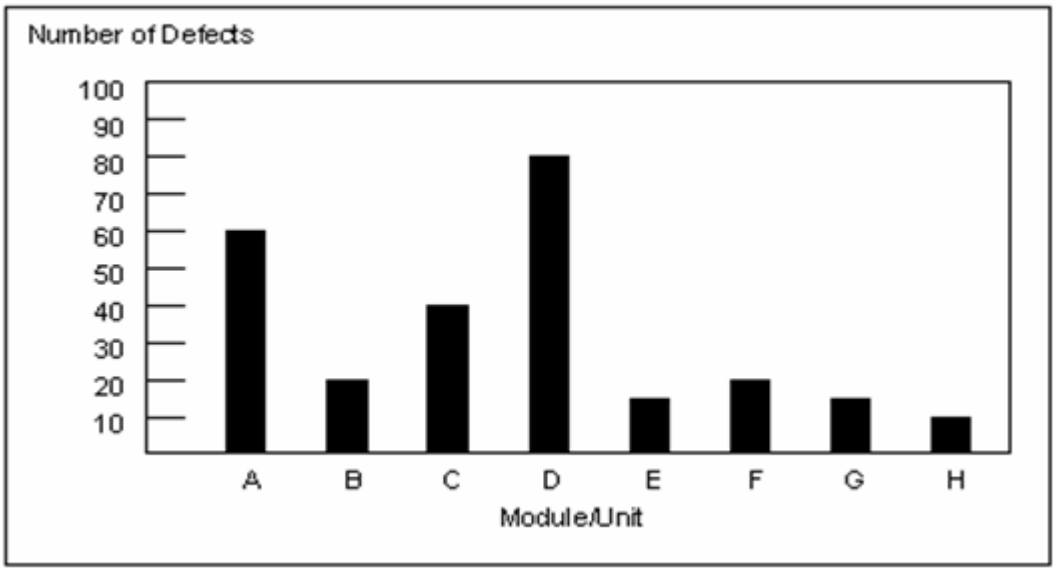


Figure 6-23 Defect Distribution Report

6.4.1.8.2 How to Interpret the Report

This report can help identify modules that have an excessive defect rate. A variation of the report could list the cumulative defects by test. For example, defects uncovered in test 1, the cumulative defects uncovered by the end of test 2, the cumulative defects uncovered by test 3, and so forth. Frequently, modules that have abnormally high defect rates are those that have ineffective architecture, and thus are candidates for rewrite rather than additional testing.

6.4.1.9 Relative Defect Distribution Report

The purpose of this report is to normalize the defect distribution presented. The normalization can be by function points or lines of code. This will permit comparison of defect density among the modules/units.

6.4.1.9.1 Report Example

The Normalized Defect Distribution Report illustrated in Figure 6-24 shows the same eight modules presented in Figure 6-23. However, in this example, the defect rates have been normalized to defects per 100 function points or defects per 1,000 lines of code, to enable the reader of the report to compare defect rates among the modules. This was not possible in Figure 6-23 because there was no size consideration. Again, a variation that shows the number of tests can be helpful in drawing conclusions.

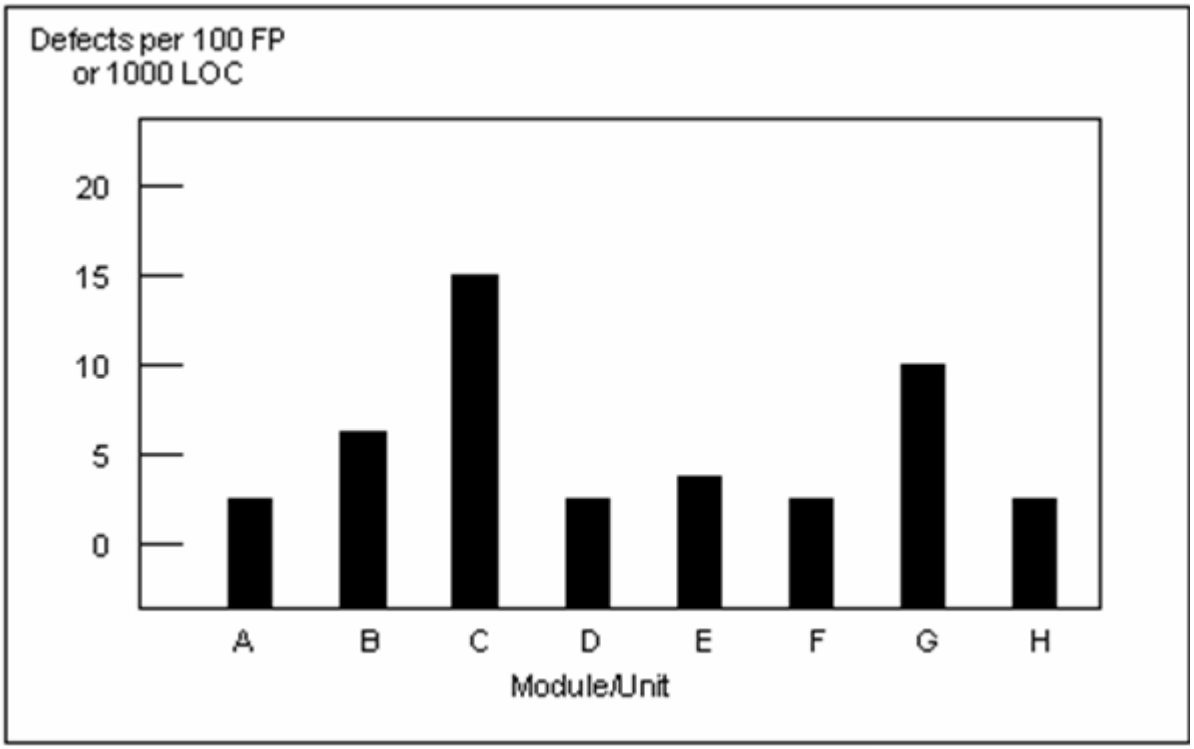


Figure 6-24 Normalized Defect Distribution Report

6.4.1.9.2 How to Interpret the Report

This report can help identify modules that have excessive defect rates. A variation of the report could show the cumulative defects by test; for example, the defects uncovered in test 1, the cumulative defects uncovered by the end of test 2, the cumulative defects uncovered by test 3, and so forth. Frequently, modules that have abnormally high defect rates are those that have ineffective architecture, and thus are candidates for rewrite rather than additional testing.

6.4.1.10 Testing Action Report

This is a summary action report prepared by the test team. It is designed for the test manager and the software development manager. The information contained in the report should be listed as necessary to the test manager and/or the development manager to properly direct the team toward a successful implementation date.

6.4.1.10.1 Report Example

The Testing Action Report example in Figure 6-25 lists four pieces of information helpful to most test managers:

- Tests Behind Schedule – Total number of tests behind schedule, meaning either they have not been performed or contain an excessive number of defects that prevent their completion on the scheduled date.

- Uncorrected Critical Defects – The total number of critical defects not yet corrected.
- Major Uncorrected Defects Over 5 Days Old – The absolute number of major defects waiting more than five days to be corrected.
- Number of Uncovered Defects Not Corrected – The total number of defects awaiting correction.

Tests Behind Schedule:
Uncorrected Critical Defects:
Major Uncorrected Defects Over 5 Days Old:
Number of Uncovered Defects Not Corrected:

Figure 6-25 Testing Action Report

These items are examples of what could be included in the Testing Action Report. Most are included in the other reports, but this report is a summation, or a substitute, for the other reports.

6.4.1.10.2 How to Interpret the Report

The test manager should carefully monitor the status of testing and take action when testing falls behind schedule.

6.4.1.11 Individual Project Component Test Results

As testing is completed on each project component, the tester should issue test reports for the individual component.

6.4.1.11.1 Report Example

An Individual Projects and Interface Report is illustrated in Figure 6-26. It describes a standard for such a report, indicating it should discuss the scope of the test, the test results, what works and does not work, and recommendations.

1. Scope of Test This section indicates which functions were and were not tested.
2. Test Results This section indicates the results of testing, including any variance between what is and what should be.
3. What Works / What Does Not Work This section defines the functions that work and do not work and the interfaces that work and do not work.
4. Recommendations This section recommends actions that should be taken to: Fix functions/interfaces that do not work. Make additional improvements.

Figure 6-26 Individual Project Component Test Report

- Scope of Test – In any report on testing it is important to show the scope, otherwise, the reader will assume that exhaustive testing has occurred, which is never the case. Testing is a risk-oriented activity in which resources should be expended to minimize the major risks. Exhaustive testing is not possible, practical or economical. Thus testing is never designed to assure that there are no defects remaining in the software and the scope will explain what the testers accomplished.
- Test Results – This is straightforward, describing the result of the testing including any variance between the expected and actual outputs.
- What Works/What Does Not Work – Where detailed interim Test Reports are available, the “what works” and “what does not work” sections may merely reference those reports or attach those reports.
- Recommendations – This section is a critical part of the report, because the reader is usually removed from the project being tested and the technical recommendations provided by the testers can help with the reader’s business decision. For example, testers may indicate that there is a 50/50 probability that the system will terminate abnormally in production due to dating problems. With that information in hand, a business decision might be made to put the software into operation, but develop effective backup recovery procedures in case the termination occurs.

6.4.1.12 Summary Project Status Report

The Summary Project Status Report is illustrated in Figure 6-27. It provides general information about software and uses graphics to summarize the status of each project component. The design of the report and use of color enables the reader to quickly and easily access project information.

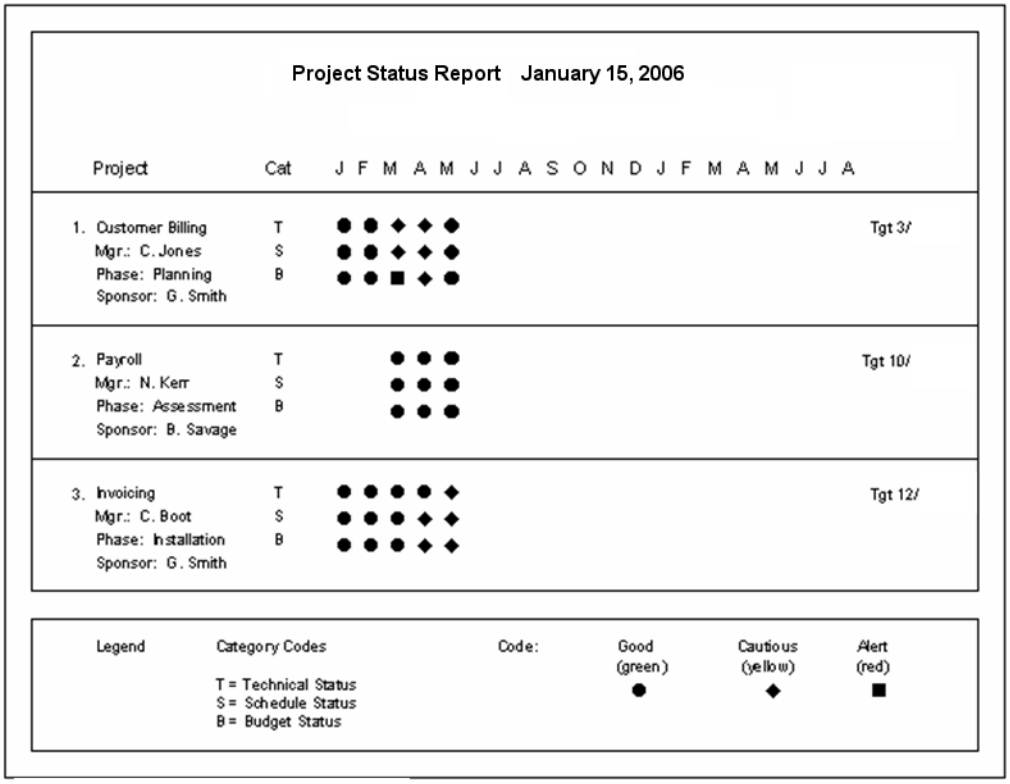


Figure 6-27 Summary Project Status Report Example

The Summary Project Status Report is divided into these four sections:

1. Report Date Information

   The date of the report should be on the report. The information that is contained in the report should be current as of that date.

2. Project Information

   Project information appears in a column on the left side of the report. Each project has its own "cell" where information about the project appears. Each cell contains the official project name, the name of the project manager, the phase of the project (e.g., planning, requirements, development, and implementation) and the name of the executive sponsor.

3. Timeline Information

   Timeline information appears in a chart that displays project status over a 16-month period. It shows project status by measuring technical, budgeting, and scheduling considerations. The year and month (abbreviated with initials) appear along the top of the chart to indicate the month-by-month status of each project.

Technical (T), Scheduling (S), and Budget (B) information also appears in the chart, and is specific to each project. These three considerations measure the status of each project:

- Technical status (T) shows the degree to which the project is expected to function within the defined technical and/or business requirements.
- Scheduling status (S) shows the degree to which the project is adhering to the current approved schedule.
- Budgeting status (B) shows the degree to which the project is adhering to the current approved budget. Expenditures for the budget include funds, human resources, and other resources.

4. Legend Information

   The report legend, which is located along the bottom of the page, defines the colors and symbols used in the report, including category and color codes. The following colors could be used to help to quickly identify project status:

   - A green circle could mean there are no major problems and that the project is expected to remain on schedule.
   - A yellow circle could indicate potentially serious deviation from project progression.
   - A red circle could mean a serious problem has occurred and will have a negative effect on project progression.

6.4.1.13 Individual Project Status Report

The Individual Project Status Report as illustrated in Figure 6-28 provides information related to a specific project component. The design of the report enables the reader to quickly and easily access project information.

6.4.1.13.1 Project Information

The project information normally includes the following six sections:

1. Name of the report
2. Date the report is issued
3. Name of the executive sponsoring the project
4. Name of the project manager
5. General project information
6. Quick-status box containing a color-coded rectangle indicating the overall status of the project

6.4.1.13.2 General Project Information

This section of the report contains general information about the project. It should include the work request number; a brief description of the project; and show the phase of the project

(e.g., planning, requirements, development, and implementation), as well as important project dates and figures, which include:

- Project start date, determined by official approval, sponsorship, and project management
- Original target date for project completion
- Current target date for project completion
- Phase start date of the current phase
- Original target date for completion of the current phase
- Current target date for completion of the current phase
- Original budget allotted for the project
- Current budget allotted for the project
- Expenses to date for the project

6.4.1.13.3 Project Activities Information

The Project Activities section of the report gives a history of the project over an extended period. The Project Activities chart measures the status according to the phase of the project. The project phases used might be:

- Planning
- Requirements
- Development
- Implementation

Comments may be added below each phase to track specific project developments or occurrences. A timeline should be included in the chart to measure each phase of the project. Color-coded circles could indicate the status of each phase.

Future activities for the project should be indicated showing the expected date of project completion, or the current target date.

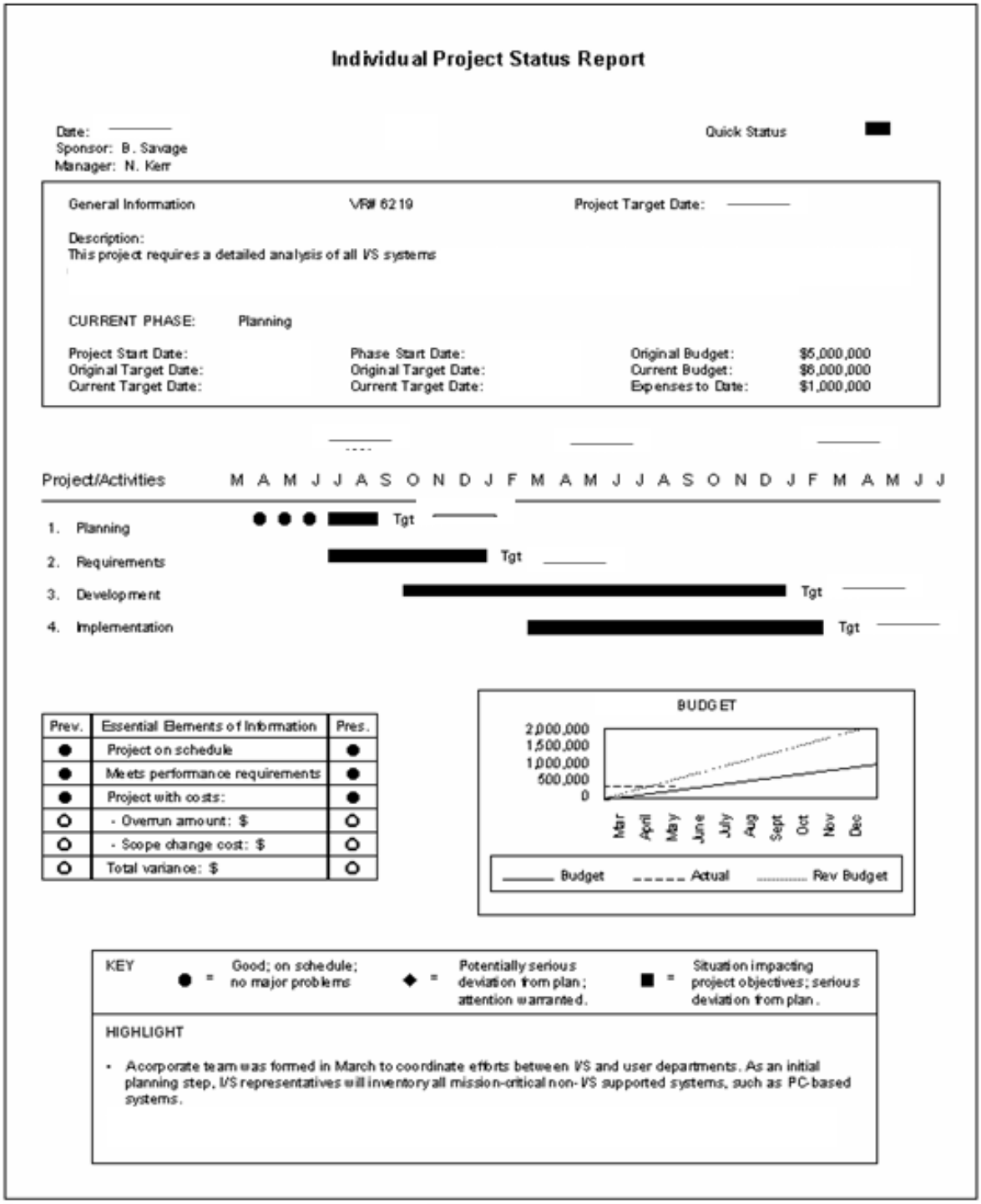


Figure 6-28 Example of an Individual Project Status Report

6.4.1.13.4 Essential Elements Information

The Essential Elements section of the report also contains a chart. It measures the status of the project by comparing it to the previous status of the project. The chart could use the color-

coded circles and list considerations that allow the reader to quickly gather project statistics. These considerations ask:

- Is the project on schedule?
- Do the current project results meet the performance requirements?
- Are the project costs within the projected budget?
- Is the project cost over-budget?
- What is the dollar amount of the project budget overrun?

These questions can be answered by comparing the previous report results to the current report results.

This section of the report also includes a graph that compares projected costs to actual costs. The projected cost line can appear in one color; the actual cost line appears in another color. This graph shows you whether the project is adhering to the current approved budget.

6.4.1.13.5 Legend Information

The report legend, which should be located along the bottom of the page, defines the colors and symbols used in the report, including category and color codes. The following symbols can be used to help to quickly identify project status:

- The ● indicates there are no major problems and that the project is expected to remain on schedule.
- The ♦ means there is a potentially serious deviation from project progression.
- The ■ indicates a serious problem has occurred and will have a negative effect on project progression.

6.4.1.13.6 Project Highlights Information

The project highlights should appear at the bottom of the report. This section may also contain comments explaining specific project developments or occurrences that affect progression.

6.4.2 Final Test Reports

Test reports should be prepared at the conclusion of each level of testing. This might include:

- Unit Test Report
- Integration Test Report
- System Test Report
- Acceptance Test Report

The test reports are designed to report the results of testing as defined in the Test Plan. Without a well-developed Test Plan, which has been executed in accordance with the plan, it is difficult to develop a meaningful test report.

All final test reports should be designed to accomplish the following three objectives:

- Define the scope of testing – this is normally a brief recap of the Test Plan
- Present the results of testing
- Draw conclusions and recommendations from those test results

The final test report may be a combination of electronic data and printed information. For example, if the Function Test Matrix is maintained electronically, there is no reason to print that, as the detail is available electronically if needed. The printed final report will summarize that data, draw the appropriate conclusions, and present recommendations.

The final test report has the following objectives:

- Inform the developers what works and what does not work.
- Provide information to the users of the software system so that they can determine whether the system is ready for production; and if so, to assess the potential consequences and initiate appropriate actions to minimize those consequences.
- After implementation, help the project trace problems in the event the application malfunctions in production. Knowing which functions have been correctly tested and which ones still contain defects can assist in taking corrective action.
- Use the test results to analyze the test process for the purpose of preventing similar defects from occurring in the future. Accumulating the results of many test reports to identify which components of the software development process that are defect-prone. These defect-prone components identify tasks or steps that, if improved, could eliminate or minimize the occurrence of high-frequency defects.

6.4.2.1 Description of Test Reports

There is no generally accepted standard regarding the type, content and frequency of test reports. However, it is reasonable to assume that some type of report should be issued after the conclusion of each test activity. This would include reports at the conclusion of these test activities:

- Unit test
- Integration test
- System test

The individual who wrote the unit normally conducts unit testing. The objective is to assure all the functions in the unit perform correctly, and the unit structure performs correctly. The report should focus on what was tested, the test results, defects uncovered and, what defects have not been corrected, plus the unit tester's recommendations as to what should be done prior to integration testing. Figure 6-29 illustrates an example of a Unit Test Report.

Tests Conducted:
Test Results:
Defects Uncovered:
Uncorrected Critical Defects:
Number of Minor Defects Not Corrected:
Recommendations:

Figure 6-29 Unit Test Report Example

6.4.2.2 Integration Test Report

Integration testing tests the interfaces between individual projects or units. A good Test Plan will identify the interfaces and institute test conditions that will validate interfaces. Given this, the integration report follows the same format as the Unit Test Report, except that the conditions tested are the interfaces.

As testing is completed on each project, integration test reports could be issued for the individual projects. Figure 6-30 is an example of such a report, indicating it should discuss the scope of the test, the test results, what works and does not work, and recommendations.

1. Scope of Test This section indicates which interfaces were and were not tested.
2. Test Results This section indicates the results of testing, including any variance between what is and what should be.
3. What Works/What Does Not Work This section defines the interfaces that work and do not work.
4. Recommendations This section recommends actions that should be taken to: Fix interfaces that do not work. Make additional improvements.

Figure 6-30 Integration Test Report Example

In any report on testing, it is important to show the scope; otherwise, the reader will assume that exhaustive testing has occurred, which is never the case. Testing is a risk-oriented activity in which resources should be expended to minimize the major risks. Exhaustive testing is neither possible, practical, nor economical. Thus, testing is never designed to assure that there are no defects remaining in the software and the scope will explain what the testers accomplished.

The remainder of the report is straightforward, describing the result of the testing – specifically, what works and what does not work in recommendations. Where detailed interim test reports are available, the "what works" and "what does not work" sections may merely reference those reports or attach those reports.

The recommendations section is a critical part of the report, because the reader is usually removed from the project being tested and the technical recommendations provided by the testers can help with the reader's business decision. For example, testers may indicate that there is a 50/50 probability that the system will terminate abnormally in production due to dating problems. A business decision might then be made to put the software into operation, but develop effective backup recovery procedures in case the termination occurs.

6.4.2.3 System Test Report

Skill Category 4, Test Planning presented a system test plan standard that identified the objectives of testing, what was to be tested, how it was to be tested, and when tests should occur. The System Test Report should present the results of executing that Test Plan. Figure 6-31 illustrates the test reporting standard that is based on the test plan standard.

1. General Information

   1.1 *Summary*. Summarize both the general functions of the software tested and the test analysis performed.

   1.2 *Environment*. Identify the software sponsor, developer, user organization, and the computer center where the software is to be installed. Assess the manner in which the test environment may be different from the operation environment, and the effects of this difference on the tests.

   1.3 *References*. List applicable references, such as:
   - 1.3.1 Project request (authorization).
   - 1.3.2 Previously published documents on the project.
   - 1.3.3 Documentation concerning related projects.
   - 1.3.4 FIPS publications and other reference documents.

2. Test Results and Findings

   Identify and present the results and findings of each test separately in paragraphs 2.1 through 2.n.

   2.1 *Test (identify)*.
   - 2.1.1 *Validation tests*. Compare the data input and output results, including the output of internally generated data, of this test with the data input and output requirements. State the findings.
   - 2.1.2 *Verification tests*. Compare what is shown on the document to what should be shown.

   2.n *Test (identify)*. Present the results and findings of the second and succeeding tests in a manner similar to that of paragraph 2.1.

3. Software Function Findings

   Identify and describe the findings on each function separately in paragraphs 3.1 through 3.n.

   3.1 *Software Function*.
   - 3.1.1 *Performance*. Describe briefly the function. Describe the software capabilities designed to satisfy this function. State the findings as to the demonstrated capabilities from one or more tests.
   - 3.1.2 *Limits*. Describe the range of data values tested. Identify the deficiencies, limitations, and constraints detected in the software during the testing with respect to this function.

   3.n. *Function (identify)*. Present the findings on the second and succeeding functions in a manner similar to that of paragraph 3.1.

Figure 6-31 System Test Report Standard Example

6.4.3 Guidelines for Report Writing

The following two guidelines are provided for writing and using the report information:

1. Develop a baseline.

   The data extracted from individual project reports can be used to develop a baseline for the enterprise based on mean scores of the reporting criteria. Rather than comparing quality, productivity, budget, defects, or other categories of metrics to external organizations, valuable management information can be made available. From this baseline, individual projects can be compared. Information from projects consistently

scoring above the enterprise baseline can be used to improve those projects that are marginal or fall below the enterprise baseline.

2. Use good report writing practices. The following are examples of good report writing:
   - Allow project team members to review the draft and make comments before the report is finalized.
   - Don’t include names or assign blame.
   - Stress quality.
   - Limit the report to two or three pages stressing important items; include other information in appendices and schedules.
   - Eliminate small problems from the report and give these directly to the project people.
   - Hand-carry the report to the project leader.
   - Offer to have the testers work with the project team to explain their findings and recommendations.

Skill Category

7

User Acceptance Testing

The objective of software development is to develop the software that meets the true needs of the user, not just the system specifications. To accomplish this, testers should work with the users early in a project to clearly define the criteria that would make the software acceptable in meeting the user needs. As much as possible, once the acceptance criterion has been established, they should integrate those criteria into all aspects of development. This same process can be used by software testers when users are unavailable for test; when diverse users use the same software; and for beta testing software. Although acceptance testing is a customer and user responsibility, testers normally help develop an acceptance test plan, include that plan in the system test plan to avoid test duplication; and, in many cases, perform or assist in performing the acceptance test.

Acceptance Testing Concepts	**7-1**
Roles and Responsibilities	**7-5**
Acceptance Test Planning	**7-7**
Acceptance Test Execution	**7-12**

7.1 Acceptance Testing Concepts

It is important that both software testers and user personnel understand the basics of acceptance testing. This section will address:

- Acceptance testing concepts
- Difference between system test and acceptance test

Acceptance testing is formal testing conducted to determine whether a software system satisfies its acceptance criteria and to enable the buyer to determine whether to accept the

system. Software acceptance testing at delivery is usually the final opportunity for the buyer to examine the software and to seek redress from the developer for insufficient or incorrect software. Frequently, the software acceptance test is the only time the buyer is involved in acceptance and the only opportunity the buyer has to identify deficiencies in a critical software system. The term critical implies economic or social catastrophe, such as loss of life; it implies the strategic importance to an organization's long-term economic welfare. The buyer is thus exposed to the considerable risk that a needed system will never operate reliably because of inadequate quality control during development. To reduce the risk of problems arising at delivery or during operation, the buyer must become involved with software acceptance early in the acquisition process.

Software acceptance is an incremental process of approving or rejecting software systems during development or maintenance, according to how well the software satisfies predefined criteria. For the purpose of software acceptance, the activities of software maintenance are assumed to share the properties of software development.

Acceptance decisions occur at pre-specified times when processes, support tools, interim documentation, segments of the software, and finally the total software system must meet predefined criteria for acceptance. Subsequent changes to the software may affect previously accepted elements. The final acceptance decision occurs with verification that the delivered documentation is adequate and consistent with the executable system and that the complete software system meets all buyer requirements. This decision is usually based on software acceptance testing. Formal final software acceptance testing must occur at the end of the development process. It consists of tests to determine whether the developed system meets predetermined functionality, performance, quality, and interface criteria. Criteria for security or safety may be mandated legally or by the nature of the system.

Acceptance testing involves procedures for identifying acceptance criteria for interim life cycle products and for accepting them. Final acceptance not only acknowledges that the entire software product adequately meets the buyer's requirements, but also acknowledges that the process of development was adequate. As a life cycle process, software acceptance enables:

- Early detection of software problems (and time for the customer or user to plan for possible late delivery)
- Preparation of appropriate test facilities
- Early consideration of the user's needs during software development
- Accountability for software acceptance belongs to the customer or user of the software, whose responsibilities are:
    - Ensure user involvement in developing system requirements and acceptance criteria
    - Identify interim and final products for acceptance, their acceptance criteria, and schedule
    - Plan how and by whom each acceptance activity will be performed
    - Plan resources for providing information on which to base acceptance decisions
    - Schedule adequate time for buyer staff to receive and examine products and evaluations prior to acceptance review

- Prepare the Acceptance Plan
- Respond to the analyses of project entities before accepting or rejecting
- Approve the various interim software products against quantified criteria at interim points
- Perform the final acceptance activities, including formal acceptance testing, at delivery
- Make an acceptance decision for each product

The customer or user must be actively involved in defining the type of information required, evaluating that information, and deciding at various points in the development activities if the products are ready for progression to the next activity.

Acceptance testing is designed to determine whether the software is fit for use. The concept of fit for use is important in both design and testing. Design must attempt to build the application to fit into the user's business process; the test process must ensure a prescribed degree of fit. Testing that concentrates on structure and requirements may fail to assess fit, and thus fail to test the value of the automated application to the business.

The four components of fit are:

1. Data

   The reliability, timeliness, consistency, and usefulness of the data included in the automated application.

2. People

   People should have the skills, training, aptitude, and desire to properly use and interact with the automated application.

3. Structure

   The structure is the proper development of application systems to optimize technology and satisfy requirements.

4. Rules

   The rules are the procedures to follow in processing the data.

The system must fit into these four components of the business environment as illustrated in Figure 7-1. If any of the components fails to fit properly, the success of the application system will be diminished. Therefore, testing must ensure that all the components are adequately prepared and developed, and that the four components fit together to provide the best possible solution to the business problem.

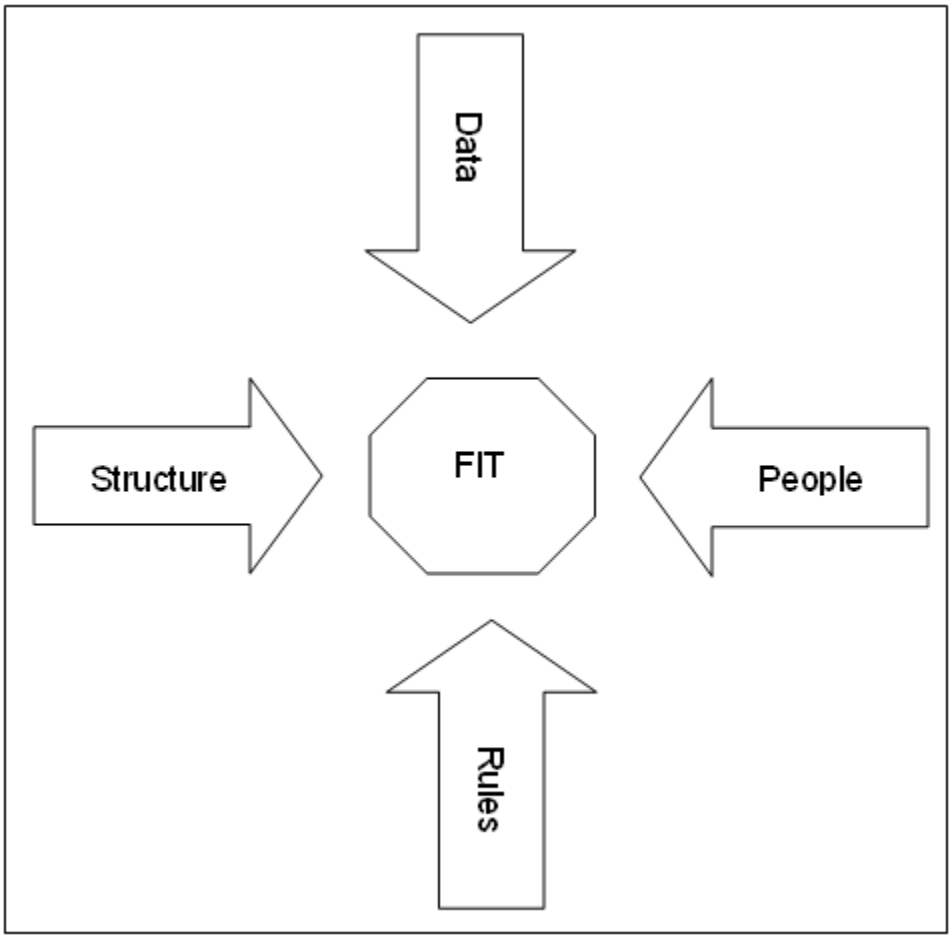


Figure 7-1 Software Change Testing and Training

The objective of acceptance testing is to determine throughout the development cycle that all aspects of the development process meet the user's needs. There are many ways to accomplish this. The user may require that the implementation plan be subject to an independent review of which the user may choose to be a part, or simply prefer to input acceptance criteria into the review process.

Acceptance testing should be an ongoing activity that tests both interim and final products. Do not wait until the end of the development process so that unnecessary time is expended making corrections that will prove unacceptable to the system user.

7.1.1 Difference between Acceptance Test and System Test

Acceptance testing is performed by user personnel and may include assistance by software testers. System testing is performed by developers and/or software testers. The objective of both types of testing is to assure that when the software is complete it will be acceptable to the user.

System test should be performed before acceptance testing. There is a logical sequence for testing, and an important reason for the logical steps of the different levels of testing. Unless each level of testing fulfills its objective, the following level of testing will have to compensate for weaknesses in testing at the previous level.

In most organization units, integration and system testing will focus on determining whether or not the software specifications have been implemented as specified. In conducting testing to meet this objective it is unimportant whether or not the software specifications are those needed by the user. The specifications should be the agreed upon specifications for the software system.

The system specifications tend to focus on the software specifications. They rarely address the processing concerns over input to the software, nor do they address the concerns over the ability of user personnel to effectively use the system in performing their day-to-day business activities.

Acceptance testing should focus on input processing, use of the software in the user organization, and whether or not the specifications meet the true processing needs of the user. Sometimes these user needs are not included in the specifications; sometimes these user needs are incorrectly specified in the software specifications; and sometimes the user was unaware that without certain attributes of the system, the system was not acceptable to the user. Examples include users not specifying the skill level of the people who will be using the system; processing may be specified but turnaround time not specified, and the user may not know that they have to specify the maintainability attributes of the software.

Effective software testers will focus on all three reasons why the software specified may not meet the user's true needs. For example they may recommend developmental reviews with users involved. Testers may ask users if the quality factors are important to them in the operational software. Testers may work with users to define acceptance criteria early in a development process so that the developers are aware and can address those acceptance criteria.

7.2 Roles and Responsibilities

The roles and responsibilities of all parties involved in acceptance testing should be incorporated into the test planning process. This section will specifically address:

- User's role
- Software tester's role

7.2.1 User's Role

The user's role in acceptance testing begins with the user making the determination as to whether acceptance testing will or will not occur. If the totality of user's needs have been incorporated into the software requirements, then the software testers should test to assure those needs are met in unit, integration, and system testing.

If acceptance testing is to occur the user has primary responsibility for planning and conducting acceptance testing. This assumes that the users have the necessary testing competency to develop and execute an acceptance test plan.

If the user does not have the needed competency to develop and execute an acceptance test plan the user will need to acquire that competency from other organizational units or out source the activity. Normally, the IT organization's software testers would assist the user in the acceptance testing process if additional competency is needed.

At a minimum the user will have the following roles in acceptance testing:

- Defining acceptance criteria in a testable format
- Providing the use cases that will be used in acceptance testing
- Training user personnel in using the new software system
- Providing the necessary resources, primarily user staff personnel, for acceptance testing
- Comparing the actual acceptance testing results with the desired acceptance testing results (NOTE: This may be performed using testing software)
- Making decisions as to whether additional work is needed prior to placing the software in operation, whether the software can be placed in operation with additional work to be done, or whether the software is fully acceptable and can be placed into production as is

If the software does not fully meet the user needs, but will be placed into operation, the user must develop a strategy to anticipate problems and pre-define the actions to be taken should those problems occur.

7.2.2 Software Tester's Role

Software testers can have one of three roles in acceptance testing. First is no involvement at all. In that instance the user accepts full responsibility for developing and executing the acceptance test plan. The second role is that of an advisor. The user will develop and execute the test plan, but rely on software testers to compensate for a lack of competency on the part of the users, or to provide a quality control role. The third role is an active participant in software testing. This role can include any or the entire acceptance testing activities.

The role of the software tester cannot include defining the acceptance criteria, or making the decision as to whether or not the software can be placed into operation. If software testers are

active participants in acceptance testing, then they may conduct any part of acceptance testing up to the point where the results of acceptance testing are documented.

A role that software testers should accept is developing the acceptance test process. This means that they will develop a process for defining acceptance criteria, develop a process for building an acceptance test plan, develop a process to execute the acceptance test plan, and develop a process for recording and presenting the results of acceptance testing.

7.3 Acceptance Test Planning

The acceptance test plan basically follows the practices used for developing an overall system test plan. Specifically this section will address:

- Acceptance Criteria
- Acceptance Test Plan
- Use Case Test Data

7.3.1 Acceptance Criteria

The user must assign the criteria the software must meet to be deemed acceptable. Ideally, this is included in the software requirements specifications. In preparation for developing the acceptance criteria, the user should:

- Acquire full knowledge of the application for which the system is intended
- Become fully acquainted with the application as it is currently implemented by the user's organization
- Understand the risks and benefits of the development methodology that is to be used in correcting the software system
- Fully understand the consequences of adding new functions to enhance the system

Acceptance requirements that a system must meet can be divided into these four categories:

- Functionality Requirements

  These requirements relate to the business rules that the system must execute.

- Performance Requirements

  These requirements relate to operational aspects, such as time or resource constraints.

- Interface Quality Requirements

  These requirements relate to connections from one component to another component of processing (e.g., human-machine, machine-module).

- Overall Software Quality Requirements

These requirements specify limits for factors or attributes such as reliability, testability, correctness, and usability.

The criterion that a requirements document may have no more than five statements with missing information is an example of quantifying the quality factor of completeness. Assessing the criticality of a system is important in determining quantitative acceptance criteria. The user should determine the degree of criticality of the requirements by the above acceptance requirements categories.

By definition, all safety criteria are critical; and by law, certain security requirements are critical. Some typical factors affecting criticality include:

- Importance of the system to organization or industry
- Consequence of failure
- Complexity of the project
- Technology risk
- Complexity of the user environment

Products or pieces of products with critical requirements do not qualify for acceptance if they do not satisfy their acceptance criteria. A product with failed noncritical requirements may qualify for acceptance, depending upon quantitative acceptance criteria for quality factors. Clearly, if a product fails a substantial number of noncritical requirements, the quality of the product is questionable.

The user has the responsibility of ensuring that acceptance criteria contain pass or fail criteria. The acceptance tester should approach testing assuming that the least acceptable corrections have been made; while the developer believes the corrected system is fully acceptable. Similarly, a contract with what could be interpreted as a range of acceptable values could result in a corrected system that might never satisfy the user's interpretation of the acceptance criteria.

For specific software systems, users must examine their projects' characteristics and criticality in order to develop expanded lists of acceptance criteria for those software systems. Some of the criteria may change according to the phase of correction for which criteria are being defined. For example, for requirements, the "testability" quality may mean that test cases can be developed automatically.

The user must also establish acceptance criteria for individual elements of a product. These criteria should be the acceptable numeric values or ranges of values. The buyer should compare the established acceptable values against the number of problems presented at acceptance time. For example, if the number of inconsistent requirements exceeds the acceptance criteria, then the requirements document should be rejected. At that time, the established procedures for iteration and change control go into effect.

Table 7-1, "Example of Required Information to Document Acceptance Criteria" explains what users need to provide to document the acceptance criteria. It should be prepared for each hardware or software project within the overall project, where the acceptance criteria requirements should be listed and uniquely numbered for control purposes.

Table 7-1: Example of Required Information to Document Acceptance Criteria

Criteria	Action
Hardware/Software Project	The name of the project being acceptance-tested. This is the name the user or customer calls the project.
Number	A sequential number identifying acceptance criteria.
Acceptance Requirement	A user requirement that will be used to determine whether the corrected hardware/software is acceptable.
Critical / Non -Critical	Indicate whether the acceptance requirement is critical, meaning that it must be met, or non-critical, meaning that it is desirable but not essential.
Test Result	Indicates after acceptance testing whether the requirement is acceptable or not acceptable, meaning that the project is rejected because it does not meet the requirement.
Comments	Clarify the criticality of the requirement; or indicate the meaning of the test result rejection. For example: The software cannot be run; or management will make a judgment after acceptance testing as to whether the project can be run.

After defining the acceptance criteria, determine whether meeting the criteria is critical to the success of the system. As shown in Table 7-2, this is indicated by placing a check mark in the Yes or No columns under Critical. Note that if an acceptance criterion were critical, then the system would not be accepted if that requirement has not been met.

Table 7-2: Acceptance Criteria Example

Number	Acceptance Requirement	Critical		Test Result		Comments
		Yes	No	Accept	Reject	
1	The system must execute to end of job.	√				Payroll will not run in a production status until this requirement has been met.
2	The results of payroll must be correct.	√				Payroll will not run in a production status until this requirement has been met.

Table 7-2 is an example of two acceptance criteria for a payroll project. It shows that both acceptance criteria are critical for the project. The Test Result column is blank, indicating the test has not yet been performed. The Comments column reports that the payroll system will not run unless these two critical requirements are met.

7.3.2 Acceptance Test Plan

The first step to achieve software acceptance is the simultaneous development of a Software Acceptance Plan, general project plans, and software requirements to ensure that user needs are represented correctly and completely. This simultaneous development will provide an overview of the acceptance activities, to ensure that resources for them are included in the project plans. Note that the initial plan may not be complete and may contain estimates that will need to be changed, as more complete project information becomes available.

Acceptance managers define the objectives of the acceptance activities and a plan for meeting them. Knowing how the software system is intended to be used in the operational environment and the risks associated with the project's life cycle provide a basis for determining these acceptance objectives. Because users may provide most of this information, initial planning sessions may be interactive between acceptance managers and users to assure that all parties fully understand what the acceptance criteria should be.

After the initial Software Acceptance Plan has been prepared, reviewed, and approved, the acceptance manager is responsible for implementing the plan and assuring that the plan's objectives are met. It may have to be revised before this assurance is warranted.

Table 7-2: lists examples of information that should be included in a Software Acceptance Plan. The first section of the plan is an overview of the software development or maintenance project, followed by major sections for management responsibilities and administrative matters. The plan's overview section describes the technical program for software acceptance. Details for each software acceptance activity or review appear in separate sections as supplements to the overview.

Project Description	Type of system; life cycle methodology; user community of delivered system; major tasks system must satisfy; major external interfaces of the system; expected normal usage; potential misuse; risks; constraints; standards and practices.
User Responsibilities	Organization and responsibilities for acceptance activities; resource and schedule requirements; facility requirements; requirements for automated support; special data, and training; standards, practices, and conventions; updates and reviews of acceptance plans and related products.
Administrative Procedures	Anomaly reports; change control; record-keeping; communication between developer and manager organizations.
Acceptance Description	Objectives for entire project; summary of acceptance criteria; major acceptance activities and reviews; information requirements; types of acceptance decisions; responsibility for acceptance decisions.

Figure 7-2 Information Included in an Acceptance Test Plan

The plan must include the techniques and tools that will be utilized in acceptance testing. Normally, testers will use the organization's current testing tools, which should be oriented toward specific testing techniques.

Two categories of testing techniques can be used in acceptance testing: structural and functional. Remember that acceptance testing must be viewed in its broadest context; not the minimal testing that some users perform after the information system professionals have concluded their testing.

The functional testing techniques help ensure that the requirements/specifications are properly satisfied by the software system. Functional testing is not concerned with how processing occurs, but with the results of processes.

Structural testing ensures sufficient checking of the implementation of the function by finding test data that will force sufficient coverage of the structured presence in the implemented software. It evaluates all aspects of this structure to verify that the structure is sound.

7.3.3 Use Case Test Data

A use case is a test case which represents how the software will be used in operation. A use case is built on a business transaction and can be test data or a test script. Unit testing will attempt to determine whether there are any variances between unit specifications and the unit as it is executed. Integration testing will attempt to determine if there is a variance between specified integration and actual integration. System testing will validate that the system, when assembled, will perform as specified. The test cases and scripts used for these three levels of testing are focused more on the components of the software than business transactions.

Many software testers do not have an adequate knowledge of the business to create business or use cases for test purposes. For example, an online data entry clerk may need to go to more than one source to gather the information needed to enter a business transaction. For example, they may need to look up the credit history of a customer prior to approving that order. While the software tester would be primarily concerned with the single system being developed, the user is concerned with the totality of events that lead to a business transaction being entered into the software.

When use cases are developed by clerical people intimately familiar with the system, they tend to know the type of problems that are typical in the business system. Thus, they can simulate those unusual events through use cases that may not be developed during normal system testing.

An individual use case consists of:

- Preconditions that set the stage for the series of events that should occur for the use case
- Post-conditions that state the expected outcomes of the above process
- Sequential narrative of the execution of the use case

See “Skill Category 5, Executing the Test Plan” for information on the process for preparing use cases.

7.4 Acceptance Test Execution

The execution of the test plan should be performed in accordance with the test plan. This section will focus on:

- Execute the Acceptance Test Plan
- Acceptance Decision

7.4.1 Execute the Acceptance Test Plan

The objective of this step is to determine whether the acceptance criteria have been met in a delivered product. This can be accomplished through reviews, which involve looking at interim products and partially developed deliverables at various points throughout the developmental process. It can also involve testing the executable software system. The determination of which (or both) of these techniques to use will depend on the criticality of the software, the size of the software program, the resources involved, and the time period over which the software is being developed.

Software acceptance criteria should be specified in the formal project plan. The plan identifies products to be tested, the specific pass/fail criteria, the reviews, and the types of testing that will occur throughout the entire life cycle.

Acceptance decisions need a framework in which to operate. Items such as contracts, acceptance criteria, and formal mechanisms are part of this framework. Software acceptance must state or refer to specific criteria that products must meet in order to be accepted. A principal means of reaching acceptance in the development of critical software systems is to hold a periodic review of interim software documentation and other software products.

A disciplined acceptance program for software of any type may include reviews as a formal mechanism. When the acceptance decision requires change, another review becomes necessary to ensure that the required changes have been properly configured and implemented and that any affected segments are acceptable. For large or complicated projects, several reviews may be necessary during the development of a single product.

Some software acceptance activities may include testing pieces of the software; formal software acceptance testing occurs at the point in the development life cycle when the user accepts or rejects the software. This means a contractual requirement between the user and the project team has been met. Rejection normally means additional work must be done on the system in order to become acceptable to the user. Final software acceptance testing is the last opportunity for the user to examine the software for functional, interface, performance, and

quality features prior to the final acceptance review. The system at this time must include the delivered software, all user documentation, and final versions of other software deliverables.

7.4.2 Acceptance Decision

Final acceptance of software based on software acceptance testing usually means that the software project has been completed, with the exception of any caveats or contingencies. Final acceptance for the software occurs, and the developer has no further development obligations (except, of course, for maintenance, which is a separate issue).

Typical acceptance decisions include:

- Required changes are accepted before progressing to the next activity
- Some changes must be made and accepted before further development of that section of the product; other changes may be made and accepted at the next major review
- Progress may continue and changes may be accepted at the next review
- No changes are required and progress may continue

The goal is to achieve and accept “perfect” software, but usually some criteria will not be completely satisfied for each product, in which case the user may choose to accept less-than-perfect software. The user must have established in advance, individual and collections of requirements.

Software acceptance is a contractual process during which users and developers identify criteria for the acceptance of software systems. Developers must agree to the users’ acceptance criteria. The users must define the acceptance criteria based on the system requirements for functionality, performance, interface quality, and overall software quality, as well as project characteristics such as the correction methodology (or variant). The buyer bases acceptance decisions on analyses and reviews of the products and on results from software product assurance activities.

The users must plan and manage the software acceptance program carefully to assure the adequate resources are available throughout the acceptance activities. Early in the process, they must include detailed plans for software acceptance testing. Such early planning enables all those involved in the software project to focus on the requirements and how well the evolving system is satisfying those requirements. Software acceptance requires adequate resources and commitment from the beginning. Its completion will result in software that delivers to its users the services they require.

This page intentionally left blank.

Skill Category

8

Testing Software Developed by Contractors

There are many challenges when testing software developed by a contractor, or an external organization. It is management's responsibility that acquired software meets the needs of their organization. Contractors will test the software they build, but that does not relieve management from their quality responsibilities. Management must put into place those test processes within their organization that provide the assurance that acquired software will perform as expected. This skill category discusses two test processes that are representative of best practices for testing acquired software are for commercial off-the shelf (COTS) software and software developed under contract by an outside organization. Executing those defined test processes should be performed by software testers.

Please note the following definitions:

- The term *contractors* will be used to mean contractors, outsourcers, offshore software developers, and developers of COTS software.
- The term *contracting organization* will be used to denote the customer, group, or organization that is purchasing the COTS or contracted software.

Challenges in Testing Acquired Software	**8-2**
COTS Software Test Process	**8-5**
Contracted Software Test Process	**8-11**

8.1 Challenges in Testing Acquired Software

There is a trend in the software industry for organizations to move from in-house developed software to COTS software and software developed by contractors. These contractors are not part of the contracting organization and are referred to as outsourcers. Contractors working in another country are referred to as offshore software developers.

There are some common differences between software developed in-house and any software developed by a contractor, and then differences specific to COTS. Quality professionals should be familiar with these differences as they impact their quality responsibilities.

Two differences between software developed in-house and software developed by contractors are:

- Relinquishment of control

  The software is developed by individuals who are not employees of the organization, and thus it is difficult to oversee the development process. The contracting organization cannot direct the employees of the other organization, nor have control over the many day-to-day decisions that are made in developing software.

- Loss of control over reallocation of resources

  If work needs to be done to correct problems and/or speed up development, the contracting organization cannot take workers off one project and assign them to another project.

8.1.1 Purchased COTS software

COTS software is normally developed prior to an organization selecting that software for its use. For smaller, less expensive software packages the software is normally “shrink wrapped” and is purchased as is. As the COTS software becomes larger and more expensive, the contracting organization may be able to specify modifications to the software.

Differences or challenges faced with testing COTS software include:

- Task or items missing
- Software fails to perform
- Extra features
- Does not meet business needs
- Does not meet operational needs
- Does not meet people needs

8.1.1.1 Evaluation versus Assessment

Many organizations select COTS software on evaluation which is a static analysis of the documentation and benefits of the software, versus performing an assessment during which the software is tested in a dynamic mode before use.

8.1.2 Contracted Software

The differences in testing software developed in-house and software developed by a contractor include the following:

- Quality factors may not be specified

  There are many factors such as reliability and ease of use which are frequently not included as part of the contractual criteria. Thus, when the software is delivered it may not be as easy to use or as reliable as desired by the contracting organization.

- Non-testable requirements and criteria

  If the requirements or contractual criteria are not in measurable and testable terms then the delivered result may not meet the intent of the contracting organization.

- contracting organization's standards may not be met

  Unless the contract specifies the operational standards and documentation standards the delivered product may be more complex to use than desired by the contracting organization.

- Missing requirements

  Unless detailed analysis and contractual specifications work is complete the contractor may realize during the development of the software that requirements are missing and thus the cost of the contract could escalate significantly.

- Overlooked changes in standards or technology

  If changes in the standards that the contracting organization must meet, or new desirable technology is incorporated into the contract after work has begun, there may be significant cost to modify the software to incorporate the new standards or technology.

- Training and deployment may be difficult

  If software is developed by a contractor there may be inadequate knowledge in the contracting organization to provide the appropriate training for staff and to ensure that deployment is effective and efficient.

8.1.2.1 Additional Differences with Contractors in another Country (Offshore)

Experience has shown that over 50% of the software developed by offshore organizations fails to meet the expectations of the contracting organization. Since many of the decisions to have software developed offshore are economic decisions, the differences associated with having

the software developed offshore negate the economic advantages in many cases. These offshore testing differences are:

- Cultural differences

  There may be a significant difference in the culture and values between the contracting organization and the offshore organization.

- Communication barriers

  The language of the offshore organization may be different or difficult to comprehend which causes difficulty in communicating the needs and desires of the contracting organization.

- Loss of employee morale and support

  Employees who would like to have developed the software may resent the software being developed offshore and thus make it difficult for the offshore developed software to be successful.

- Root cause of the contractor IT organization not addressed

  Frequently, an offshore organization is chosen because there are problems in the contracting organization that executives do not want to address. For example, the problems might include a lack of training for the employees in the contracting organization or other perhaps better options for software development were not explored.

The above discussions are not meant to be an exhaustive list of the differences between in-house developed software and software developed by a contractor. The objective is so the software tester recognizes some potential root causes of software quality. If those differences are not adequately addressed in the contract or through additional test activities, the probability of the contracted or offshore-developed software failing to meet the needs of the acquiring organization increases.

8.1.2.2 Software Tester's Responsibility for Software Developed by a Contractor

While the software may be developed by a contractor, the responsibility for the quality of that software cannot be contracted. The contracting organization is still responsible for the quality of the organization. There must be a process to monitor the development and validate the correct functioning of the software when it is developed by a contractor.

The software tester is the individual who should accept the responsibility for software testing developed by a contractor. This may mean that the software tester needs to periodically visit the contractor during the entire developmental period of the software to ensure the quality. Many of the same practices used to test in-house developed software are applicable to software developed by a contractor. For example, conducting reviews at specific checkpoints should occur; Acceptance testing should be conducted on all software regardless of how developed.

The software tester's specific responsibility for software developed by a contractor could include assuring that the process for selecting COTS software and contracting with a contractor are adequate.

The software tester needs to look at how the contractor tests in relation to the SEI CMMI® Capability Maturity Model. If testing is done at a Level 1 maturity there will be great variability and thus many disappointments in the delivered product and services. On the other hand, as those test processes move to a Level 5 maturity, the probability of getting exactly what is wanted from COTS software and contracted software is very high.

8.2 COTS Software Test Process

COTS software will have been developed and tested by the contractor, and be ready to put into operation when the contracting organization begins its software testing activities. The amount of testing to be performed on COTS software will be determined by the risk associated with placing that software into operation. If the risk is minimal, a decision may be made not to test. If the risk is high, extensive testing may be required.

The following seven step test process is designed to test high-risk COTS software. As the risk decreases so should the test effort. Reducing the test effort can be accomplished by eliminating all or parts of these seven steps that are discussed further:

- Assure completeness of needs specification
- Define critical success factor
- Determine compatibility with your computer environment
- Assure the software can be integrated into your business system work flow
- Demonstrate the software in operation
- Evaluate the people fit
- Acceptance test the COTS software

8.2.1 Assure Completeness of Needs Specification

This step determines whether you have adequately defined your needs. Your needs should be defined in terms of the following two categories of outputs:

1. Output products and reports

   Output products and reports are specific documents that you want produced by the computer system. In many instances, such as printing payroll checks for example, the style and format of these output products is important. This does not mean that the specific location of the check has to be defined but, rather, the categories of information to be included on the check. Computer-produced reports may also be important for tax information (e.g., employee withholding forms sent to governmental units), financial

statements where specific statements are wanted (e.g., balance sheets or statements of income and expense), or contracting organization invoice and billing forms which you might want preprinted to include your logo and conditions of payment.

2. Management decision information

   This category tries to define the information needed for decision-making purposes. In the computer product/report category you were looking for a document; in this case you are looking for information. How that information is provided is unimportant. Thus, the structure of the document, what the documents are, or their size, frequency, or volume are not significant. All you need is information.

8.2.1.1 Define Critical Success Factor

This step determines whether the software package will be successful in meeting your business needs. Critical Success Factors (CSFs) are those criteria or factors that must be present in the acquired software for it to be successful. A question that is often asked is, “Are the needs are the same as the critical success factors?”. They are, but needs are not defined in a manner that makes them testable, and they may be incomplete.

Often the needs do not take into account some of the intangible criteria that make the difference between success and failure. In other words, the needs define what we are looking for, and the critical success factors tell us how we will evaluate that product after we get it. They are closely related and complementary, but different in scope and purpose. The following list helps to illustrate the differences by using the needs and requirements for the automobile, and then the CSFs on which the automobile will be evaluated:

- Automobile requirements and needs: seats six people, has four doors, has a five-year guarantee on motor, gets 20 miles per gallon or greater, and costs under $12,000.
- Critical success factors: operates at 20.5 cents or less per mile, experiences no more than one failure per year, maintains its appearance without showing signs of wear for two years.

Use some of these more common CSFs when testing COTS software:

- Ease of use – the software is understandable and usable by the average person.
- Expandability – the vendor plans to add additional features in the future.
- Maintainability – the vendor will provide support/assistance to help utilize the package in the event of problems.
- Cost-effectiveness – the software package makes money for your business by reducing costs, and so on.
- Transferability – if you change your computer equipment the vendor indicates that they will support new models or hardware.
- Reliability – in computer language, the system is friendly, meaning that it will help you get your transactions entered into the system so that you can produce your results readily.

- Security – the system has adequate safeguards to protect the data against damage (for example, power failures, operator errors, or other situations that could cause you to lose your data).

8.2.1.2 Determine Compatibility with Your Computer Environment

This is not a complex step. It involves a simple matching between your processing capabilities and limitations, and what the vendor of the software says is necessary to run the software package. The most difficult part of this evaluation is ensuring the multiple software packages can properly interface.

This step is best performed by preparing a checklist defining your compatibility needs. Software vendors are generally good about identifying the needed hardware and operating system compatibility.

In addition to the hardware on which the software runs, and the operating system with which it must interact to run, there are two other important compatibilities:

- Compatibility with other software packages (software vendors are generally not good in identifying compatibility with other software packages)
- Compatibility with available data

If you have no other software packages that you want to have interact with this one, or no data on computer-readable media, you need not worry about these aspects of compatibility. However, as you do more with your computer these aspects of compatibility will become more important, and the hardware and operating compatibility will become routine and easy to verify.

Systems compatibility is defined in data processing jargon as interoperability. This term refers to the amount of effort required to intercouple or interconnect computer systems. In other words, how do you tie two or more programs together so that they will work and pass data between them? For example, if you have a payroll system it may be desirable to pass that payroll summary information to your general-ledger system. The ability to pass information from system to system is an extremely important part of data processing. Much of the success of the Lotus Corporation was based in its ability to intercouple five office software functions so that information could be readily passed from one to another.

To help assure compatibility, prepare a list with the information described below. The list is divided into hardware, operating systems, programs, and data.

8.2.1.2.1 Hardware Compatibility

List the following characteristics for your computer hardware:

- Hardware vendor(s)
- Amount of main storage
- Disk storage unit identifier
- Disk storage unit capacity

- Type of printer(s)
- Number of print columns
- Type of terminal(s)
- Maximum terminal display size
- Keyboard restrictions

8.2.1.2.2 Operating Systems Compatibility

For the operating system used by your computer hardware, list:

- Name of operating system (e.g., UNIX or Windows)
- Version of operating system in use

8.2.1.2.3 Software Compatibility

List all of the programs with which you expect or would like this specific application to interact. Be sure that you have the name of the vendor and, if applicable, the version of the program. Note that as discussed earlier, this linkage may only be verifiable by actually attempting to interact two or more systems using common data.

- Data Compatibility

  In many cases, software compatibility will answer the questions on data compatibility. However, if you created special files you may need descriptions of the individual data elements and files. Again, as with software compatibility, you may have to actually verify through trial whether the data can be read and used by other software.

8.2.1.3 Assure the Software can be Integrated into Your Business System Work Flow

Each computer system makes certain assumptions. Unfortunately, these assumptions are rarely stated in the vendor literature. The danger is that you may be required to do some manual processing functions that you may not want to do in order to utilize the software.

The objective of this step is to determine whether you can implement the COTS into your existing work environment without disrupting your entire operation. Remember that:

- Your current system is based on a certain set of assumptions
- Your current system uses existing forms, existing data, and existing procedures
- The COTS system is based on a set of assumptions
- The COTS system uses a predetermined set of forms and procedures
- Your current system and the COTS system may be incompatible
- If they are incompatible, the COTS system is not going to change—you will have to
- You may not want to change – then what?

The objective of this process is to illustrate the type and frequency of work flow changes that will be occurring. You can see what will happen when the COTS system is brought into your

organization. For example, there might be tasks performed now that weren't performed before, or tasks that were previously performed but are no longer necessary, or tasks which had been performed by people which will now be performed by the COTS system. Having the COTS system perform those tasks might mean that the oversight that people had been giving will not be available any more.

At the end of this test, you will need to decide whether you are pleased with the revised work flow. If you feel the changes can be effectively integrated into your work flow, the potential COTS system has passed the test. If you feel the changes in work flow will be disruptive, you may want to fail the COTS in this test and either look for other software or continue your current processing.

If the testing is to continue, you should prepare a clean data flow diagram indicating what actions need to be taken to integrate the COTS system into your organization's work flow. This new data flow diagram becomes your installation plan of action. It will tell you what changes need to be made, who is involved in them, what training might be necessary, and areas of potential work flow problems.

8.2.1.4 Demonstrate the Software in Operation

This step analyzes the many facets of software. Software developers are always excited when their program goes to what they call "end of job." This means that it executes and concludes without abnormally terminating (i.e., stops after doing all the desired tasks). While this is one aspect of the demonstration, observing the functioning of software is like taking an automobile for a test drive. The more rigorous the test, the greater the assurance you are getting what you expect.

Demonstrations can be performed in either of the following ways:

- Computer store–controlled demonstration

  In this mode, the demonstration is conducted at the computer store, by computer store personnel, using their data. The objective is to show you various aspects of the computer software, but not to let you get too involved in the process. This is done primarily to limit the time involved in the demonstration.

- contracting organization site demonstration

  In this mode, the demonstration takes place at your site, under your control, by your personnel, using your information. It is by far the most desirable of all demonstrations, but many software COTS computer stores may not permit it unless you purchase the COTS.

These aspects of computer software should be observed during the demonstration:

- Understandability

  As you watch and listen to the demonstration, you need to evaluate the ease with which the operating process can be learned. If the commands and processes appear more like magic than logical steps, you should be concerned about implementing the concept in your organization. If you have trouble figuring out how to do it, think about how difficult it may

be for some of your clerical personnel who understand neither the business application nor the computer.

- Clarity of communication

Much of the computer process is communication between man and machine. That is, you must learn the language of the computer software programs in order to communicate with the computer. Communication occurs through a series of questions and responses. If you do not understand the communications, you will have difficulty using the routine.

- Ease of use of instruction manual

While monitoring the use of the equipment, the tasks being demonstrated should be cross-referenced to the instruction manual. Can you identify the steps performed during the demonstration with the same steps included in the manual? In other words, does the operator have to know more than is included in the manual, or are the steps to use the process laid out so clearly in the manual that they appear easy to follow?

- Functionality of the software

Ask to observe the more common functions included in the software: Are these functions described in the manual? Are these the functions that the salesperson described to you? Are they the functions that you expected? Concentrate extensively on the applicability of those functions to your business problem.

- Knowledge to execute

An earlier test has already determined the extent of the salesperson's knowledge. During the demonstration, you should evaluate whether a lesser-skilled person could as easily operate the system with some minimal training. Probe the demonstrator about how frequently they run the demonstration and how knowledgeable they are about the software.

- Effectiveness of help routines

Help routines are designed to get you out of trouble when you get into it. For example, if you are not sure how something works you can type the word "help" or an equivalent and the screen should provide you additional information. Even without typing "help" it should be easy to work through the routines from the information displayed on the screen. Examine the instructions and evaluate whether you believe you could have operated the system based on the normal instructions. Then ask the operator periodically to call the help routines to determine their clarity.

- Evaluate program compatibility

If you have programs you need to interact with, attempt to have that interaction demonstrated. If you purchased other software from the same store where you are now getting the demonstration, they should be able to show you how data is passed between the programs.

- Data compatibility

Take one of your data files with you. Ask the demonstrator to use your file as part of the software demonstration. This will determine the ease with which existing business data can be used with the new software.

- Smell test

While watching the demonstration, let part of your mind be a casual overseer of the entire process. Attempt to get a feel for what is happening and how that might impact your business. You want to end up being able to assess whether you feel good about the software. If you have concerns, attempt to articulate them to the demonstrator as well as possible to determine how the demonstrator responds and addresses those concerns.

To determine whether an individual has the appropriate skill level to use the software it is recommended to involve one or more typical users of the software in software demonstrations.

8.2.1.5 Evaluate the People Fit

The objective of this step is to determine whether your employees can use the software. This step evaluates whether employees possess the skills necessary to effectively use computers in their day-to-day work. The evaluation can be of current skills, or the program that will be put into place to teach individuals the necessary skills. Note that this includes the owner or president of the organization as well as the lowest-level employee in the organization.

The test is performed by selecting a representative sample of the people who will use the software. The sample need not be large. This group is given training that may only involve handing someone the manuals and software. The users will then attempt to use the software for the purpose for which it was intended. The results of this test will show:

- The software can be used as is
- Additional training and support is necessary
- The software is not usable with the skill sets of the proposed users

8.2.1.6 Acceptance Test the COTS Software

There is little difference between acceptance testing in-house developed software and acceptance testing acquired software. Acceptance testing is a user responsibility. Refer to Skill Category 7 for information on the acceptance testing process.

8.3 Contracted Software Test Process

If developing software under contract allowed the acquiring organization's testers to fully test the software, the testers would follow the same process as used for in-house developed software. However, the acquiring organization's testers are limited in the amount and timing of test by contractual provisions. For example, the contract may not allow reviews during development by the acquiring organizational personnel, and the developer may not release source code for test purposes.

Good test practices for software development involve the acquiring organization's testers in all phases of development and operation. Therefore, the following software tester responsibilities are identified for the best practices test process:

- Assure the process for contracting software is adequate
- Assure that requirements and contract criteria are testable
- Review the adequacy of the contractors test plan
- Perform acceptance testing on the software
- Issue a report on the adequacy of the software to meet the needs of the organization
- Ensure knowledge transfer occurs, and intellectual property rights are protected
- Incorporate copyrighted material into the contracting organization's manuals
- Assure the ongoing operation and maintenance of the contracted software
- Assure the effectiveness of contractual relations

8.3.1 Assure the Process for Contracting Software is Adequate

Without a process for contracting for software those processes would be subject to great variability. One contract may work well, while other outside developed software contracts may result in failure.

Testers are not lawyers, but they can test to determine whether or not the contracting process includes addressing good contracting practices. The following is a guide for testers on what should be addressed during the contracting process.

Contracts are legal documents. To fully understand the impact of provisions being included in, or excluded from, the contract may require legal training. However, the following information should be included in all contracts:

- What is done.

  The contract should specify the deliverables to be obtained as a result of execution of the contract. Deliverables should be specified in sufficient detail so that it can be determined whether or not the desired product has been received.

- Who does it.

  The obligations of both contractual parties should be spelled out in detail in the contract.

- When it is done.

  The dates on which the contractual obligations need to be filled should be specified in the contract.

- How it is done.

  The contract should specify the methods by which the deliverables are to be prepared if that is important in achieving the contractual needs. For example, the contracting organization may not want certain types of source instructions used in developing an

application system because they plan to perform the maintenance with the in-house personnel.

- Where it is done.

  The location where the deliverables are to be prepared, delivered, operated, and maintained should be specified as part of the contract.

- Penalties for nonperformance.

  If the contractual agreements are not followed, the penalties should be specified in the contract. For example, if the contractor is late in delivering the work products, the contract may specify a penalty of x dollars per day.

The concerns that need to be addressed by a contracting process include the following factors:

- Warranty

  The guarantees provided by the contractor that the deliverables will meet the specifications. This segment of the contract should state specifically what the contractor warrants, and what the contractor will do if the deliverables fail to meet the warranty guarantees. For example, if the contractor guarantees the application will provide a one-second response, and the implemented application fails to meet those specifications, the contract defines recourse against the contractor.

- Deliverables

  The application system, documentation, and other products to be provided by the contractor should be described in great detail. For example, a phrase like “provide for adequate controls” is a meaningless phrase in that the deliverables are not measurable. The contractor’s definition of “adequate controls” may be entirely different than that of the contracting organization, but loose wording such as this, except in cases of gross negligence, eliminates recourse. The product specifications should include as much detail as practical, and as much as necessary to ensure that the organization gets the product they want.

- Delivery date

  The date on which the product is to be delivered should be specified in the contract. This may be multiple dates, in that a product may be delivered for testing and then another date specified for when those corrections will be made, etc.

- Commencement date

  The date at which the contract becomes effective should be specified in the contract. This is particularly important if delivery dates are keyed to the commencement date, such as; a deliverable will be available sixty days after the contract is signed.

- Installation

  The contractor’s and contracting organization’s commitment for installing the application should be specified. If the contractor is to have personnel present to help during the installation, that should be specified. If the contractor is to provide machine time on certain days, that, too, should be specified in the contract.

- Updates

  The type of continual maintenance provided by the contractor for the application system should be specified. This is particularly important if the contracting organization operates in an environment where operating systems are regularly changed. The contract might specify that the contractor will provide necessary updates so that the system will operate on new versions of the operating system being used by the contracting organization.

- Contractor support

  The types, quantity, and location of contractor support should be specified. In addition, some contracting organizations specify the caliber of people that should provide that support. For example, systems analysts should have a minimum of five years' programming and systems experience, and at least one year experience with the application system. It is also important to specify where the support will occur. For example, is contractor support to be provided at the contracting organization's place of business, or must the contracting organization's personnel go to the contractor's place of business to get that support.

- Costs

  The amounts to be paid to the contractor by the contracting organization should be specified, including payment terms. If there are penalty clauses for late payments, that should be specified, as well as the rights of the contracting organization to withhold payments if the contractor fails to provide individual deliverables or other support. Ideally, the value for each deliverable should be specified so that the amounts withheld are determined by contract for failure to perform. The more precise the description, the fewer the problems.

- Foreign attachments

  If the application system is interconnected with applications and/or other software from different contractors, that interrelationship should be specified. It is important to state in the contract who has the primary responsibility for tracing and correcting errors. In a multi-contractor environment, when no contractor accepts primary responsibility for error tracking, the contracting organization may need to expend large amounts of funds to trace errors because of the unwillingness of contractors to accept this responsibility.

- Penalties

  Penalties assessed in the event of failure on the part of either the contractor or the contracting organization to meet contractual obligations should be covered in the contract. Where dollar penalties are to be assessed, the amount should be specified in the contract. For example, if the contractor is late in delivering the product, a per-day dollar penalty can be assessed, as well as a dollar penalty for failure of the contracting organization to make computer time available, etc.

- Life of contract

The duration of the contract should be specified, including any rights to continue or discontinue the contract at the end of its life. For example, the contracting organization may have the right to extend the contract another year for X dollars.

- Modification capability

  The contract should specify what type of modifications (to the application system) the contractor is willing to make. This should include how promptly modifications will be made, and the costs and other terms associated with making those modifications. The contract should also state what type of modifications can be made by the contracting organization, and which ones must be made by the contractor.

- Service discontinuance

  If the contractor decides to discontinue service on the application system, the contracting organization's rights in those instances should be specified. For example, if any of the training material or application system is copyrighted then those copyrights should be passed to the contracting organization in the event that service on the purchased products is discontinued.

- Manual/training discontinuance

  If the contractor decides to discontinue training manuals, service manuals, or training courses, the rights to that material should revert to the contracting organization. If this is not included in the contract, the contracting organization may be prevented from using material and training courses copyrighted by the contractor, without making additional payments.

- Acceptance test criteria

  The contract should specify the criteria which determine that the delivered product is acceptable. The acceptance test criteria should not only cover the delivered application system, but any documentation and training material to be included with the application system. The contract should also state where and by whom the acceptance test is to be performed.

- Purchase versus lease

  The options of the contracting organization to either purchase or lease the application system should be outlined in the contract. If it is a lease contract, the rights of the contracting organization to purchase that application, if available, should be specified in the contract.

- Fairness of contract

  Both the contracting organization and the contractor should want a contract that is fair to both parties. If the contract is extremely harsh, one or the other parties may find it more desirable to terminate than to continue the contract. For example, if the penalty clauses to the contractor are extremely harsh, and the contractor finds the effort to prepare the deliverables is greater than anticipated, it may be more advantageous to the contractor to terminate the contract than to be late and pay the unrealistic penalty amounts. Thus, an unfair contract may not achieve the desired objectives on the part of either or both parties.

- Performance of maintenance

  The location, method, and timing of the performance of maintenance should be specified in the contract. If it is important for the contracting organization that the maintenance be performed at the contracting organization's place of business, that needs to be specified in the contract. If not, the contractor can provide maintenance at the contractor's convenience as opposed to the contracting organization's convenience.

- Contractor training

  The type, frequency, caliber of instructors, and location of contractor training for the application system should be included in the contract.

- Contractor manuals

  The manuals needed to operate and maintain the application system should be specified in the contract. These normally include computer operator manuals, user manuals, learning manuals, and systems documentation manuals. The contract should be as specific as possible regarding the size, content, method of presentation, and continued maintenance of the material in the manual.

- Supplies

  The types of supplies provided by the contractor should be specified in the contract. This may include input forms, printer forms, and computer media. The cost, availability, and rights of the contracting organization to reproduce any of the supplies should the contractor be unable to make delivery, should be included in the contract.

- Transportability

  The rights of the contracting organization to move the application system from location to location should be stated in the contract. Transportability should also include the rights of the contracting organization to run the application system in more than one location. There may or may not be fees associated with the rights of the contracting organization to move, duplicate, and run the application system in multiple locations.

- Termination

  In the event the contractor or contracting organization wishes to terminate the contract, the methods of terminating the contract should be specified. The termination clause should cover both cost and the return of deliverables provided under the contract. Providing for the termination can avoid a lot of bad feelings if it becomes necessary to end the contract. It also lets the organization know in advance the type of financial commitments that are associated with the termination.

- Contractor employee rights

  Contractor personnel may need to visit the premises of the contracting organization to perform service on the application system. In providing this service, the contractor needs to know if they have free access to the contracting organization's place of business, if they can use the contracting organization's equipment for testing (with or

without charge), whether or not they can store books, manuals, supplies, in the contractor's place of business, etc. Also, if the contract is tied to usage, the contractor may wish the right to examine logs and other evidence of usage.

- Governing law

  The state under which the rules of the contract are binding should be defined. It is also the laws of that state under which any dispute would be tried in court.

- Contractor inspection

  The right of the contractor to look at the records of the contracting organization should be stated. This would be necessary only in a lease arrangement if the rental is tied to usage, revenue, or other criteria based on records maintained by the contracting organization. In order to be assured that the contractor receives full and fair rental, the contractor may wish the right to examine the contracting organization's records. Contracts usually indicate where these are available, and the procedures necessary to obtain them.

- Access to developers and development documentation

  What documentation testers need, and the test activities they desire to perform need to be included in the contract.

- Assure that requirements and contract criteria are testable.

  A common term used for contracting today is performance-based contracting. This means that the performance criteria of the contractor will be defined and once defined can be monitored and measured.

It is important that all the contract criteria and software requirements that are incorporated in a contract for software development, be testable. That means as much as possible an objective measurement can be made as to whether or not that requirement/criteria has or has not been met. For example, easy-to-use criteria might specify the type and number of help screens included within the software.

As a general rule if a requirement or criteria is not testable, the contractor has too much discretion on how that requirement or criteria is implemented. It also limits the contracting organization's ability to have corrections made without additional cost for a requirement or criteria that does not meet the contracting organization's need. If it is a testable requirement, it is easy to demonstrate that the contractor has failed to meet the component of the contract.

8.3.2 Review the Adequacy of the Contractor's Test Plan

This responsibility is to evaluate the adequacy and completeness of testing that will be performed. It is important for the software tester to develop an opinion on the contractor's ability to deliver software that meets the requirements or criteria in the contract. It can also provide insight on the ability of the contractor to develop software at a reasonable cost. For example if the test plan indicates that the defect removal efficiency at the requirements phase is 95%, then the quality professional knows that only 5% of the requirement defects will move

to the design phase. On the other hand, if the contractor does not begin testing until after the software is compiled, then there may be extensive rework and potential delays in getting the software on the scheduled date.

The software tester should look for the type of comprehensive test plan that is included in Skill Category 4. The extensiveness of that test plan would provide the software tester with reasonable assurance that the delivered software will meet the contractual criteria.

8.3.3 Assure Development is Effective and Efficient

Verification is a much more effective means for identifying defects than validation. However to perform verification practices during development the acquiring organization must have the ability to interact with developers and have access to the developer documentation. To do this the contract must allow for verification activities. Skill Category 1 includes the types of verification activities testers could perform on software being developed under contract.

8.3.4 Perform Acceptance Testing on the Software

The software tester should not allow contracted software to be placed into operation without some acceptance testing.

The software tester may or may not be involved in the actual acceptance testing. However, the quality professional must determine that an approved acceptance test plan will be put into place to test software developed by outside organizations. Refer to Skill Category 7 for what should be included in a user acceptance test plan.

It has been demonstrated extensively that the cost of not acceptance testing is normally much greater than the cost of acceptance testing. It only takes a few problems in acquired software to far exceed the cost of acceptance testing.

The extent of acceptance testing software will be dependant upon the risk associated with the use of that software. As the use and importance diminishes so does the amount of acceptance testing. Likewise, the greater assurance the software tester has in the ability of a contractor of software to produce high quality, defect-free software the fewer acceptance testings that need to be conducted. At a minimum the acceptance testing should validate that:

- The documentation is consistent with the software execution
- The documentation is understandable
- Users will be adequately trained in the software prior to use of the software
- It is operable within the operational constraints of the organization (For example, all the features and the platform that are needed to operate the software are in fact there.)

8.3.5 Issue a Report on the Adequacy of the Software to Meet the Needs of the Organization

The software tester should prepare a report on the adequacy of the software to meet the needs of the organization. The report should focus on two main issues. Does the software meet the contractual requirements? Second, does it meet the needs of the organization?

The report should focus on the two definitions of quality. These are:

- Meets requirements
- Fit for use

For example as previously stated off-shore developed software fails to meet the needs of the organization more than 50% of the time. However, these off-shore developed software may meet the specifications of the contract, but not the true needs of the contracting organization.

This report is normally prepared after acceptance testing, if software is contracted for, and the software tester has access to the developers, several reports can be prepared. For example it would discuss the quality of the software at specific checkpoints. If there are problems, the report might also address the probability of those problems being resolved during the remainder of developmental time.

8.3.6 Ensure Knowledge Transfer Occurs and Intellectual Property Rights are Protected

There are two concerns that the software tester must assure about software developed by outside organizations. The first is that there is adequate knowledge transfer about the software from the developer of the software to the acquirer. The second is that the intellectual property rights of both the contractor and contracting organization are protected.

The amount of knowledge transfer that will occur will be dependant upon the purchase/ contractual arrangements for acquiring the software. For example, contractors may not release source code to the contracting organization. The importance of this issue can change based on the relationship with the developer and/or whether the developer stays in business. For example, if the vendor goes out of business does the contracting organization then have the right to obtain the source code so the software can be maintained even though the vendor of the software is no longer in business?

Among the items that may be important in knowledge transfer are:

- Training programs for the contracting organization staff
- Being advised of defects uncovered by other organizations using the software
- Ability to contact a contractor help desk to resolve problems/get additional information

There are legal aspects of protecting the intellectual property acquired from a contractor. The contracting organization may not have the right to reproduce and distribute the software without additional compensation to the contractor.

The contracting organization may have to exercise reasonable care to protect the rights of the contractor, such as securing and providing adequate protection over the software and associated documentation.

8.3.7 Incorporate Copyrighted Material into the Contractor's Manuals

The contracting organization may also have intellectual property that they want protected. For example, they may share with the contractor proprietary material that they do not want the contractor to use for any other purpose.

In some instances, the contracting organization wants access to the software, or developer materials prior to acquiring or contracting for software. This and other aspects of protecting intellectual property may be covered in a nondisclosure agreement between the contractor and the contracting organization.

8.3.8 Assure the Ongoing Operation and Maintenance of the Contracted Software

The contractual arrangements determine the ongoing relationship between the contractor and the contracting organization. This relationship may continue as long as the contracting organization continues to use the application system. It encompasses continued service and maintenance. However, the ongoing relationship may only involve guarantee and warranty of the product.

Frequently, organizations overlook contractual agreements after the application system has gone operational. This is because problems may not occur initially, and when they do occur, the organization neglects to go back to the contract to determine the obligation of the contractor for these problems.

The major concern during the operation and maintenance of a purchased or leased application system is that both parties to the agreement comply with the contractual requirements. Contracts should be periodically reviewed to verify that the contractual requirements are being met.

Software testers can evaluate negotiations over time to determine that the contractor fulfills their part of the contract. There are also instances where the contracting organization is obligated to meet ongoing contractual obligations and compliance to those obligations should be verified.

The major concerns during the operation and maintenance of a purchased application include:

- Adequacy of control

  Controls provided in the purchased application should be sufficient to assure that the data processed by the application is accurate, complete, and authorized. Controls should provide sufficient preventive, detective, and corrective controls to enable the organization to be assured of processing integrity. Available review checklists, such as the ones provided in previous sections of this manual, provide guidance for reviewers in making this determination.

- Adequacy of documentation

  The application system should be maintainable and operable with the documentation provided by the contractor. If the organization is dependent upon the contractor for help in the day-to-day operations, the documentation is probably inadequate. When the user finds they cannot adequately operate or maintain the system, they should request more documentation from the contractor.

- Speed of service

  Service should be provided by the contractor on a basis such that the operations of the organization are not seriously curtailed. In some instances, this may mean service within a few hours, while in other instances several days may be adequate.

- Nearness of service

  The contractor should have people located such that they can service the application system in a reasonable period of time. Remote service that will be made available quickly may be adequate to satisfy this need.

- Competency of service

  The service personnel of the contractor should be sufficiently skilled to perform the tasks for which they are assigned.

- Adequacy of hardware

  The contracting organization should provide sufficient hardware so that the purchased application can run in an efficient and economical mode.

- Skilled personnel

  The contracting organization personnel should be adequately trained in the operation of the application so that they are self-sufficient.

- Multi-contractor problem resolution

  If application systems are provided by more than one contractor, procedures should be established and agreed upon by all of the contractors as to who has the primary problem resolution responsibility.

- Cost of services

  The services to be provided by the contractor during the life of the contract should be specified in terms of cost. The contracting organization should be able to anticipate the approximate cost of required service.

- Cost of operations

  If the application system is leased, and the cost of the lease is tied to usage, the cost associated with usage should be easily measurable.

- Error diagnosis

  The responsibility to diagnose problems should be documented. Those responsible should do the error diagnosis. If the contracting organization's personnel have that responsibility, they should be sufficiently trained and have sufficient aids provided by the contractor so that they can perform this function. If the contractor personnel accept that responsibility, they must be responsive to the needs of the contracting organization in error detection and correction.

- Error documentation

  Procedures should be established to specify the type of documentation collected at the time of errors. This should be collected for two purposes: first, to aid contractor personnel in further diagnosis and correction of the problem; and second, as possible recourse against the contractor for recovery of fees due to extra expenses incurred. The contractor should agree that the type of documentation being collected is sufficient for their error diagnosis and correction purposes.

8.3.9 Assure the Effectiveness of Contractual Relations

The relationship between the contractor and the contracting organization is an ongoing relationship. Time and effort must be expended to keep that a viable and healthy relationship. The relationship should not be considered fixed at the point in time the contract was signed but, rather, a continual evolving relationship in which both the interest of the contractor and the contracting organization are protected.

The contractor is anxious to sell more applications and service to the contracting organization. Therefore, special needs and interests of the contracting organization are normally handled even if they are above and beyond the contractual negotiations. These are normally performed in an effort to continually improve the relationship in hopes of ongoing business.

The contracting organization hopefully has received a valuable product from the contractor. In most instances, the contracting organization either could not produce this product, or produce it at an equivalent cost or time span. Thus, it is normally within the best interest of the contracting organization to gain more products of a similar nature.

The concerns that arise in maintaining a relationship of harmony and good that the testers could evaluate will include:

- Contractor obligations met

  The contractor should meet their requirements as specified in the contract.

- contracting organization obligations met

The contracting organization should meet their requirements as specified in the contract.

- Needs met

  It is to the benefit of both parties to have the contracting organization satisfied with the application system. Even when the initial deliverables meet the contracting organization's need, there will be ongoing maintenance required to meet the continually evolving needs of the contracting organization. The methods of doing this should be specified in the contract, and those requirements should form the basis for both parties specifying and delivering new contractual obligations.

- Limits on cost increases

  The cost specified in the contract should include provisions for ongoing costs. In an inflationary economy, it may be advantageous to have limits placed on cost increases. For example, if service is provided at an hourly rate, the increases in that rate might be specified in the contract.

- Exercising options (e.g., added features)

  Many contracts contain options for additional features or work. When new requirements are needed, it should first be determined if they can be obtained by exercising some of the options already available in the contract.

- Renegotiation

  Many contracts contain provisions to renegotiate in the event of some specified circumstances. For example, if the contracting organization wants to extend the contract, that extension may involve a renegotiation of the terms of the contract. The renegotiation process should be conducted in accordance with the contractual specifications.

- Compensation for error

  If the contractor agrees to compensate for problems due to contractor causes, the penalties should be specified in the contract.

- Returns on termination

  If the contract is terminated, the contractual termination procedures should be performed in accordance with the contract requirements.

This page intentionally left blank.

Skill Category

9

Testing Software Controls and the Adequacy of Security Procedures

A key issue for software testing is testing internal control. Security is a component of internal control that warrants special attention of testers. Interest in internal control has been highlighted by publicized penetrations of security and the increased importance of information systems and the data contained by those systems. In the U.S., the passage of the Sarbanes-Oxley Act in particular, highlighted interest in internal control. The Sarbanes-Oxley Act, sometimes referred to as *SOX*, was passed in response to the numerous accounting scandals such as Enron and WorldCom. While much of the act relates to financial controls, there is a major section relating to internal controls. For Securities and Exchange Commission (SEC)-regulated corporations, both the CEO and the CFO must personally attest to the adequacy of their organization's system of internal control. Because misleading attestation statements is a criminal offense, top corporate executives take internal control as a very important topic. Many of those controls are incorporated into information systems, and thus there is a need for testing those controls.

Several sections of this skill category discuss the processes used to develop software and, although testers are not responsible for the development process, the more you know about the process used to develop software, the more effective the testing process will become.

Principles and Concepts of Internal Control	**9-2**
Internal Control Models	**9-17**
Testing Internal Controls	**9-23**
Testing Security Controls	**9-27**

9.1 Principles and Concepts of Internal Control

There are many definitions of internal control. Most of those definitions were developed by accountants. Some of those definitions focus more on financial controls, but others take a much broader view of internal control. Note that security is part of the system of internal control.

In the 1990s, five major accounting organizations developed a framework for internal control. The five members of the group known as the Committee of Sponsoring Organizations, frequently referred to as COSO, include: Financial Executives International, American Institute of Certified Public Accountants, American Accounting Association, The Institute of Internal Auditors, and the Institute of Management Accountants.

The COSO Internal Control Framework has been widely accepted after the passage of the Sarbanes-Oxley Act. This is because the Act requires organizations to have a "framework for internal control" and the SEC, which oversees the Sarbanes-Oxley Act, only recognizes the COSO Internal Control Framework.

There is no one generally accepted definition of internal control. Many have developed definitions, some broad, some very specific. However, it is important to have a clear definition of internal control.

COSO defines internal control as:

" … A process, effected by an organization's Board of Directors, management and other personnel, designed to provide reasonable assurance regarding the achievement of objectives in the following categories:

- Effectiveness and efficiency of operations
- Reliability of financial reporting
- Compliance with applicable laws and regulations."

The following four key terms are used extensively in internal control and security:

- Risk – The probability that an undesirable event will occur.
- Exposure – The amount of loss that might occur if an undesirable event occurs.
- Threat – A specific event that might cause an undesirable event to occur.
- Control – Anything that will reduce the impact of risk.

Let's look at an example of these terms using a homeowner's insurance policy and focus on one risk, which is the risk of fire. The exposure associated with a risk of fire would be the value of your home. A threat that might cause that risk to turn into a loss might be an improper electrical connection or children playing with matches. Controls that would minimize the loss associated with that risk would include such things as fire extinguishers, sprinkler systems, fire alarms, and non-combustible material used in construction.

In looking at IT, we might look at the risk of someone penetrating a banking system and improperly transferring funds to the perpetrators personal account. The risk is the loss of

funds in the account that was penetrated. The exposure is the amount of money in the account (or the amount of money that the bank allows to be transferred electronically). The threat is inadequate security systems, which allow the perpetrator to penetrate the banking system. Controls can include limiting access to accounts with passwords, limiting the amount that can be transferred at any one time, monitoring unusual transactions such as transfers of funds to an overseas account, and controlling who can transfer money from the account.

9.1.1 Internal Control Responsibilities

While everyone in an organization, particularly Management, has some responsibility for internal control, it is the chief executive officer that holds ultimately responsibility for the organization's internal control system. Financial and accounting officers are central to the way management exercises control. All management personnel play important roles and are accountable for controlling their units' activities.

Internal auditors contribute to the ongoing evaluation of the internal control system, but they do not have primary responsibility for establishing or maintaining it. The Board of Directors and its audit committee provide important oversight to the internal control system. A number of other parties, such as lawyers and external auditors, contribute to the achievement of the organization's objectives and provide information useful in improving internal control. However, they are not responsible for the effectiveness of, nor are they a part of, the organization's internal control system.

9.1.2 Software Tester's Internal Control Responsibilities

Software systems are controlled by a system of controls embedded in the software. This section will describe that control system. Since the software system incorporates controls, software testers should test that those controls exist and perform as specified.

9.1.3 Internal Auditor's Internal Control Responsibilities

Internal auditors directly examine internal controls and recommend improvements. The Institute of Internal Auditors, the professional association representing internal auditors worldwide, defines internal auditing as:

> "… an independent, objective assurance and consulting activity designed to add value and improve an organization's operations. It helps an organization accomplish its objectives by bringing a systematic, disciplined approach to evaluate and improve the effectiveness of risk management, control, and governance processes."

The International Standards for the Professional Practice of Internal Auditing, established by the Institute of Internal Auditors, specify that internal auditors should:

- Assist the organization by identifying and evaluating significant exposures to risk and contributing to the improvement of risk management and control systems
- Monitor and evaluate the effectiveness of the organization's risk management system
- Evaluate risk exposures relating to the organization's governance, operations, and information systems regarding the:
  - Reliability and integrity of financial and operational information
  - Effectiveness and efficiency of operations
  - Safeguarding of assets
  - Compliance with laws, regulations, and contracts
- Assist the organization in maintaining effective controls by evaluating their effectiveness and efficiency and by promoting continuous improvement

All activities within an organization are potentially within the scope of the internal auditors' responsibility. In some entities, the internal audit function is heavily involved with controls over operations. For example, internal auditors may periodically monitor production quality, test the timeliness of shipments to customers or evaluate the efficiency of the plant layout. In other entities, the internal audit function may focus primarily on compliance or financial reporting-related activities.

The Institute of Internal Auditors standards also set forth the internal auditors' responsibility for the roles they may be assigned. Those standards, among other things, state that internal auditors should be independent of the activities they audit. They possess, or should possess, such independence through their position and authority within the organization and through recognition of their objectivity.

Organizational position and authority involve such matters as a reporting line to an individual who has sufficient authority to ensure appropriate audit coverage, consideration and response; selection and dismissal of the director of internal auditing only with Board of Directors or audit committee concurrence; internal auditor access to the Board or audit committee; and internal auditor authority to follow up on findings and recommendations.

Internal auditors are objective when not placed in a position of subordinating their judgment on audit matters to that of others. The primary protection for this objectivity is appropriate internal audit staff assignments. These assignments should be made to avoid potential and actual conflicts of interest and bias. Staff assignments should be rotated periodically and internal auditors should not assume operating responsibilities. Similarly, they should not be assigned to audit activities with which they were involved in connection with prior operating assignments.

It should be recognized that the internal audit function does not – as some people believe – have primary responsibility for establishing or maintaining the internal control system. That, as noted, is the responsibility of the CEO, along with key managers with designated responsibilities. The internal auditors play an important role in evaluating the effectiveness of control systems and thus contribute to the ongoing effectiveness of those systems.

9.1.4 Risk versus Control

From an academic perspective, the sole purpose of control is to reduce risk. Therefore, if there is no risk, there is no need for control. The formula for risk is as follows:

Risk = Frequency x Probable Loss

To calculate the loss due to risk, one must first determine:

- The frequency with which an unfavorable event will occur
- The probable loss associated with that unfavorable occurrence

Let's look at a simple example. There is a risk that products shipped will not be invoiced. If we were to assume that an average of two products will be shipped per day and not be invoiced and the average billing per invoice is $500, then the risk associated with not invoicing shipments is $1,000 per day.

Management has chosen to use a positive concept in addressing risk, rather than a negative concept. In other words, they recognize that there will be a risk that products will be shipped but not invoiced. To address risk such as this, management has chosen to define control objectives rather than risks.

In our shipped but not billed risk example, management would define a control objective of "All products shipped should be invoiced". They would then implement controls to accomplish that positive control objective.

9.1.5 Environmental versus Transaction Processing Controls

It is important for the quality professional to know that there are two components of controls. The first is environmental (sometimes called general controls), and the second is the transaction processing controls within an individual business application.

9.1.5.1 Environmental or General Controls

Environmental controls are the means by which management uses to manage the organization. They include such things as:

- Organizational policies
- Organizational structure in place to perform work
- Method of hiring, training, supervising and evaluating personnel
- Processes provided to personnel to perform their day-to-day work activities, such as a system development methodology for building and testing software systems.

Auditors state that without strong environmental controls the transaction processing controls may not be effective. For example, if passwords needed to access computer systems are not adequately protected the password system will not work. Individuals will either protect or not protect their password based on environmental controls such as the attention management

pays to password protection, the monitoring of the use of passwords that exist, and management's actions regarding individual workers failure to protect passwords.

Two examples of management controls are the review and approval of a new system and limiting computer room access.

- Review and Approval of a New System

  This control should be exercised to ensure management properly reviews and approves new IT systems and conversion plans. This review team examines requests for action, arrives at decisions, resolves conflicts, and monitors the development and implementation of system projects. It also oversees user performance to determine whether objectives and benefits agreed to at the beginning of a system development project are realized.

  The team should establish guidelines for developing and implementing system projects and define appropriate documentation for management summaries. They should review procedures at important decision points in the development and implementation process.

- Limiting Access to Computer Resources

  Management controls involve limiting access to computer resources. It is necessary to segregate the functions of systems analysts, programmers, and computer operators. Systems analysts and programmers should not have physical access to the operating programs, and the computer files. Use of production files should be restricted to computer operating personnel. Such a restriction safeguards assets by making the manipulation of files and programs difficult. For example, assume a bank's programmer has programmed the demand deposit application for the bank. With his knowledge of the program, access to the files in the computer room on which information about the demand depositors is contained may allow him to manipulate the account balances of the bank's depositors (including his own balance if he is a depositor).

9.1.6 Transaction Processing Controls

The object of a system of internal control in a business application is to minimize business risks. Risks are the probability that some unfavorable event may occur during processing. Controls are the totality of means used to minimize those business risks.

There are two systems in every business application. As illustrated in Figure 9-1, the first is the system that processes business transactions, and the second is the system that controls the processing of business transactions. From the perspective of the system designer, these two are designed and implemented as one system. For example, edits that determine the validity of input are included in the part of the system in which transactions are entered. However, those edits are part of the system that controls the processing of business transactions.

Because these two systems are designed as a single system, most software quality analysts do not conceptualize the two systems. Adding to the difficulty is that the system documentation is not divided into the system that processes transactions and the system that controls the processing of transactions.

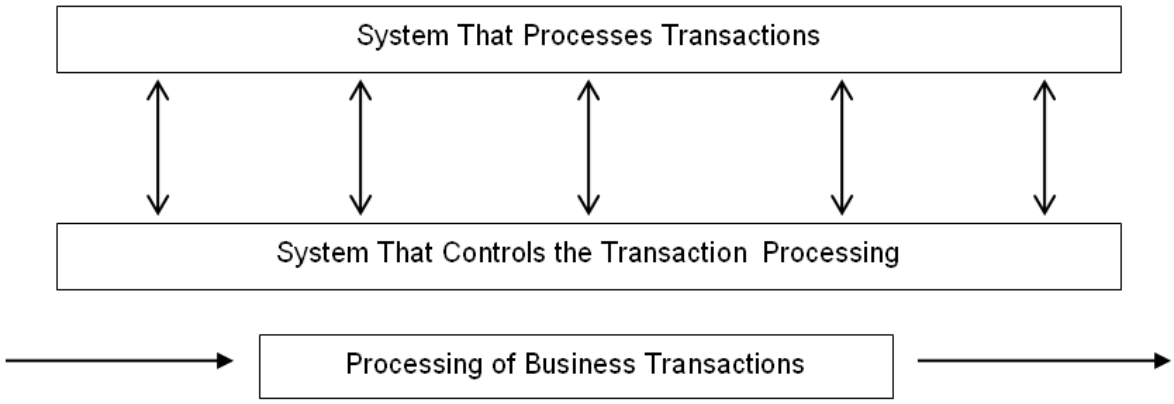


Figure 9-1 The Two Systems in Every Business Application

When one visualizes a single system, one has difficulty in visualizing the total system of internal control. For example, if one looks at edits of input data by themselves, it is difficult to see how the totality of control over the processing of a transaction is controlled. For example, there is a risk that invalid transactions will be processed. This risk occurs throughout the system and not just during the editing of data. When the system of internal controls is viewed it must address all of the risks of invalid processing from the point that a transaction is entered into the system to the point that the output deliverable is used for business purposes.

A point to keep in mind when designing transaction processing controls is that some input errors may be acceptable if they do not cause an interruption in the processing run. A simple example of this would be a misspelled description of an item. In deciding on controls, it is necessary to compare the cost of correcting an error to the consequences of accepting it. Such trade-offs must be determined for each application. Unfortunately there are no universal guidelines available.

It is important that the responsibility for control over transaction processing be separated as follows:

- Initiation and authorization of a transaction
- Recording of the transaction
- Custody of the resultant asset

In addition to safeguarding assets, this division of responsibilities provides for the efficiencies derived from specialization, makes possible a cross-check that promotes accuracy without duplication or wasted effort, and enhances the effectiveness of a management control system.

9.1.7 Preventive, Detective and Corrective Controls

This section describes three different categories of transaction processing controls, preventive, detective, and corrective and provides examples of those types of controls. Also provided is a detailed process to follow when building controls within an information system. While this activity falls outside the scope of a testing, knowing how the software is designed can greatly improve your ability to design appropriate test plans and processes.

The objectives of transaction processing controls are to prevent, detect, or correct incorrect processing. Preventive controls will stop incorrect processing from occurring; detective controls identify incorrect processing; and corrective controls correct incorrect processing. Since the potential for errors is always assumed to exist, the objectives of transaction processing controls will be summarized in five positive statements:

- Assure that all authorized transactions are completely processed once and only once.
- Assure that transaction data is complete and accurate.
- Assure that transaction processing is correct and appropriate to the circumstances.
- Assure that processing results are utilized for the intended benefits.
- Assure that the application can continue to function.

In most instances controls can be related to multiple exposures. A single control can also fulfill multiple control objectives. For these reasons transaction processing controls have been classified according to whether they prevent, detect, or correct causes of exposure. The controls listed in the next sections are not meant to be exhaustive but, rather, representative of these types of controls.

9.1.7.1 Preventive Controls

Preventive controls act as a guide to help things happen as they should. This type of control is most desirable because it stops problems from occurring. Computer application systems designers put their control emphasis on preventive controls. It is more economical and better for human relations to prevent a problem from occurring than to detect and correct the problem after it has occurred.

Preventive controls include standards, training, segregation of duties, authorization, forms design, pre-numbered forms, documentation, passwords, consistency of operations, etc.

One question that may be raised is, “At what point in the processing flow is it most desirable to exercise computer data edits?” The answer to this question is simply, “As soon as possible, in order to uncover problems early and avoid unnecessary computer processing.” Some input controls depend on access to master files and so must be timed to coincide with file availability. However, many input validation tests may be performed independently of the master files. Preferably, these tests should be performed in a separate edit run at the beginning of the computer processing. Normally, the input validation tests are included in programs to perform data-conversion operations such as transferring data files from one application to

another. By including the tests in programs performing such operations, the controls may be employed without significantly increasing the computer run time.

Preventive controls are located throughout the entire IT system. Many of these controls are executed prior to the data entering the computer programs. The following preventive controls will be discussed in this section:

- Source-data authorization
- Data input
- Source-data preparation
- Turn-around documents
- Pre-numbered forms
- Input validation
- Computer updating of files
- Controls over processing

9.1.7.1.1 Source-Data Authorization

Once data has been recorded properly, there should be control techniques to ensure that the source data has been authorized. Typically, authorization should be given for source data such as credit terms, prices, discounts, commission rates, overtime hours, and so forth.

The input documents, where possible, should have evidence of authorization and should be reviewed by the internal control group in data processing. To the extent practical, the computer should be utilized as much as possible to authorize input. This may be done through programmed controls.

9.1.7.1.2 Data Input

Data input is the process of converting data in non-machine-readable form (such as hard-copy source documents) into a machine-readable form so that the computer can update files with the transactions. Since the data input process is typically a manual operation, control is needed to ensure that the data input has been performed accurately.

9.1.7.1.3 Source-Data Preparation

In many automated systems, conventional source documents are still used and, therefore, no new control problems are presented prior to the conversion of source documents into machine-readable form. Specially designed forms promote the accuracy of the initial recording of the data. A pre-audit of the source documents by knowledgeable personnel to detect misspellings, invalid codes, unreasonable amounts, and other improper data helps to promote the accuracy of input preparation.

In IT systems where the source document is eliminated or is in a form which does not permit human review, control over source-data preparation should be such that access to, and use of, the recording and transmitting equipment is properly controlled to exclude unauthorized or improper use.

9.1.7.1.4 Turn-Around Document

Other control techniques to promote the accuracy of input preparation include the use of turn-around documents which are designed to eliminate all or part of the data to be recorded at the source. A good example of a turn-around document is the bill which you may receive from a utility company. Normally the bill has two parts: one part is torn off and included with the remittance you send back to the utility company as payment for your bill; the other you keep for your records. The part you send back normally includes pre-recorded data for your account number and the amount billed so that this returned part can be used as the input medium for computer processing of the cash receipts for the utility company.

9.1.7.1.5 Pre-Numbered Forms

Sequential numbering of the input transaction form with full accountability at the point of document origin is another traditional control technique. This can be done by using pre-numbered forms or by having the computer issue sequential numbers.

9.1.7.1.6 Input Validation

An important segment of input processing is the validation of the input itself. This is an extremely important process because it is really the last point in the input preparation where errors can be detected before files are updated. The primary control techniques used to validate the data are associated with the editing capabilities of the computer. Because of the characteristics of the computer, an IT system has unusual capabilities to examine or edit each element of information processed by it. This editing involves the ability to inspect and accept (or reject) transactions according to validity or reasonableness of quantities, amounts, codes, and other data contained in input records. The editing ability of the computer can be used to detect errors in input preparation that have not been detected by other control techniques discussed previously.

The editing ability of the computer is achieved by installing checks in the program of instructions, hence the term program checks. They include:

- Validity tests

  Validity tests are used to ensure that transactions contain valid transaction codes, valid characters, and valid field size. For example, in an accounts receivable system, if only input coded PB through PL were valid transaction codes, then input with other codes would be rejected by the computer. In a labor data collection system, all time transactions and job transactions could be checked by the computer against the random-access file of active job numbers, and non-matches indicated on a report to the shop foreman.

- Completeness tests

  Completeness checks are made to ensure that the input has the prescribed amount of data in all data fields. For example, a particular payroll application requires that each new employee hired have a unique User ID and password. A check may also be included to see that all characters in a field are either numeric or alphabetic.

- Logical tests

  Logical checks are used in transactions where various portions, or fields, of the record bear some logical relationship to one another. A computer program can check these logical relationships to reject combinations that are erroneous even though the individual values are acceptable.

- Limit tests

  Limit tests are used to test record fields to see whether certain predetermined limits have been exceeded. Generally, reasonable time, price, and volume conditions can be associated with a business event. For example, on one payroll application, the computer is programmed to reject all payroll rate changes greater than 15 percent of the old rate. The labor hour's field is checked to see if the number of hours worked exceeds 44. In another application, an exception report is generated when a customer's receivable balance plus the total of his unfilled orders exceeds his credit limit.

- Self-checking digits

  Self-checking digits are used to ensure the accuracy of identification numbers such as account numbers. A check digit is determined by performing some arithmetic operation on the identification number itself. The arithmetic operation is formed in such a way that typical errors encountered in transcribing a number (such as transposing two digits) will be detected.

- Control totals

  Control totals serve as a check on the completeness of the transaction being processed and ensure that all transactions have been transmitted properly from the source to the data processing center. Control totals are normally obtained from batches of input data. The computer can be programmed to accumulate control totals internally and make a comparison with those provided as input.

9.1.7.1.7 Computer Updating of Files

The updating phase of the processing cycle entails the computer updating files with the validated transactions. Normally computer updating involves sequencing transactions, comparing transaction records with master-file records, computations, and manipulating and reformatting data, for the purpose of updating master files and producing output data for distribution to user departments for subsequent computerized processing.

Another control technique for the proper updating of files is file maintenance. File maintenance consists of those procedures involved in making changes to the permanent-type information contained in master files such as name, address, employee number, and pay rate information in a payroll file. Since this data is so important to the proper computerized processing of files, formalized procedures are required to make changes to this type of permanent information. All master file changes should be authorized in writing by the department initiating the change. A notice or register of all changes should be furnished to the initiating department to verify that the changes were made.

9.1.7.1.8 Controls over Processing

When we discussed input validation, we saw that programmed controls are a very important part of application control. Programmed controls in computer updating of files are also very important since they are designed to detect loss of data, check arithmetic computation, and ensure the proper posting of transactions.

Three examples of programmed controls are:

- A control total is made from amount or quantity fields in a group of records and is used to check against a control established in previous or subsequent manual or computer processing.
- A hash total is another form of control total made from data in a non-quantity field (such as vendor number or customer number) in a group of records.
- Programmed checks of arithmetic calculations include limit checks, cross-footing balance checks, and overflow tests.

Let us examine some of these programmed controls.

Programmed checks to detect loss or non-processing of data are record counts, control totals and hash totals. A record count is the number of records processed by the computer. The resulting total can then be compared with a predetermined count. Normally a record count is established when the file is assembled, and the record count is carried as a control total at the end of the file or reel and is adjusted whenever records are added or deleted. For example, a record count may be established for all new hiring's or terminations processed. This record count can then be compared internally or manually to predetermined totals of new hiring's or terminations. Each time the file is processed, the records are recounted and the quantity is balanced to the original or adjusted total. Although the record count is useful as a proof of processing accuracy, it is difficult to determine the cause of error if the counts are out of balance.

Some calculations produce illogical results such as million-dollar payroll checks or negative payroll checks. Such calculations can be highlighted in exception reports with the use of limit checks, which test the results of a calculation against predetermined limits. For example, a payroll system may include limit checks to exclude, from machine payroll check preparation, all employees with payroll amounts greater than $1000 or less than $0.

Cross-footing balance checks can be programmed so that totals can be printed out and compared manually or totals can be compared internally during processing. For example, a computer-audit program is used in testing accounts receivable and in selecting accounts for confirmation. Each account is aged according to the following categories: current, 30, 60, and 90 days. The aged amounts for each account are temporarily stored in accumulators in the central processing unit. When all open items for the account have been aged, the aged totals for the account are compared to the account balance stored elsewhere in the central processing unit. Any difference, results in an error indication. The program also includes the

accumulation and printout of aged amounts for all accounts which can be manually compared with the total accounts receivable balance.

The overflow test is a widely used test to determine whether the size of a result of a computation exceeds the registered size allocated to hold it. If so, there must be a means of saving the overflow portion of the results which would otherwise be lost. Overflow control may be programmed or may be available as a hardware or software control provided by the equipment manufacturer.

Programmed checks for proper postings may be classified as file checks. Basically, these are controls used to ensure that the correct files and records are processed together. The problem of using the correct file is a significant one in IT systems because of the absence of visible records and because of the ease with which wrong information can be written on magnetic tapes and disks. The increase in the size and complexity of modern data processing systems has resulted in the growth of large system libraries containing data that can cost thousands of dollars to generate. For the purpose of preserving the integrity of data, various labeling techniques have been devised to provide maximum protection for a file to prevent accidental destruction or erasure and to ensure proper posting, updating, and maintenance. Two types of labels are used, external and internal.

External labels are a physical safeguard which properly falls under the category of documentation and operating practices. They are attached to the exterior data processing media.

9.1.7.2 Detective Controls

Detective controls alert individuals involved in a process so that they are aware of a problem. Detective controls should bring potential problems to the attention of individuals so that action can be taken. One example of a detective control is a listing of all paychecks for individuals who worked over 80 hours in a week. Such a transaction may be correct, or it may be a systems error, or even fraud.

Detective controls will not prevent problems from occurring, but rather will point out a problem once it has occurred. Examples of detective controls are batch control documents, batch serial numbers, clearing accounts, labeling, and so forth.

The following detective controls will be discussed here:

- Data transmission
- Control totals
- Control register
- Documentation and testing
- Output Checks

9.1.7.2.1 Data Transmission

Once the source data has been prepared, properly authorized, and converted to machine-processable form, the data usually is transmitted from the source department to the data processing center. Data transmission can be made by conventional means (such as messenger and mail) or by data transmission devices which allow data transmission from remote locations on a much timelier basis.

One important control technique in data transmission is batching, the grouping of a large number of transactions into small groups. Batching typically is related more to sequential-processing systems where transactions have to be put into the same order as the master files; however, batching may also apply to many direct-access systems where it may be desirable to batch input for control purposes.

Let us examine a payroll application as an illustration of batching. In such an example, the source document may include time cards (source-data preparation) which should have been approved by a foreman (data authorization). For batching, these data time cards could be divided into groups of 25, with a control total for hours worked developed for each batch along with the total for all batches. Each batch transaction and its control totals could then be sent (data transmission) to the internal control group in the IT department for reconciliation with their batch control totals. Thus batching and control totals are useful techniques for the control of both data conversion and data transmission. These control totals could also be used during the computer-processing phase where the payroll files would be updated.

Control totals should be developed on important fields of data on each record to ensure that all records have been transmitted properly from the source to the data processing center. Controls might be developed on the number of records in each batch or could be based on some quantitative field of data such as invoice amount or hours worked, etc. Such controls serve as a check on the completeness of the transaction being processed and ensure that all transactions have been received in the data processing center.

9.1.7.2.2 Control Register

Another technique to ensure the transmission of data is the recording of control totals in a log so that the input processing control group can reconcile the input controls with any control totals generated in subsequent computer processing.

9.1.7.2.3 Control Totals

Control totals are normally obtained from batches of input data. These control totals are prepared manually, prior to processing, and then are incorporated as input to the computer-processing phase. The computer can be programmed to accumulate control totals internally and make a comparison with those provided as input. A message confirming the comparison should be printed out, even if the comparison did not disclose an error. These messages are then reviewed by the internal processing control group.

9.1.7.2.4 Documentation and Testing

Accuracy of programming is ensured by proper documentation and extensive program testing procedures. Good documentation will aid in locating programming errors and will facilitate correction even in the absence of the original designer or programmer. Extensive program testing under real-life conditions, and testing all possible exceptions without actual programmer involvement, will minimize possibilities of hidden program bugs and facilitate a smooth running system.

9.1.7.2.5 Output Checks

The output checks consist of procedures and control techniques to:

- Reconcile output data, particularly control totals, with previously established control totals developed in the input phase of the processing cycle
- Review output data for reasonableness and proper format
- Control input data rejected by the computer during processing and distribute the rejected data to appropriate personnel
- Distribute output reports to user departments on a timely basis

Proper input controls and file-updating controls should give a high degree of assurance that the computer output generated by the processing is correct. However, it is still useful to have certain output controls to achieve the control objectives associated with the processing cycle. Basically, the function of output controls is to determine that the processing does not include any unauthorized alterations by the computer operations section and that the data is substantially correct and reasonable. The most basic output control is the comparison of control totals on the final output with original input control totals such as record counts or financial totals. Systematic sampling of individual items affords another output control. The testing can be done by the originating group or the control group.

One of the biggest controls in any system occurs when the originating group reviews reports and output data and takes corrective action. Review normally consists of a search for unusual or abnormal items. The programmed controls discussed above, coupled with exception reporting, actually enhance the ability of responsible personnel to take necessary corrective action.

Another form of output control in some organizations is the periodic and systematic review of reports and output data by internal audit staff. This group normally has the responsibility to evaluate operating activities of the company, including computer operations, to determine that internal policies and procedures are being followed.

9.1.7.2.6 Corrective Controls

Corrective controls assist individuals in the investigation and correction of causes of risk exposures that have been detected. These controls primarily collect evidence that can be utilized in determining why a particular problem has occurred. Corrective action is often a difficult and time-consuming process; however, it is important because it is the prime means

of isolating system problems. Many system improvements are initiated by individuals taking corrective actions on problems.

It should be noted that the corrective process itself is subject to error. Many major problems have occurred in organizations because corrective action was not taken on detected problems. Therefore detective control should be applied to corrective controls. Examples of corrective controls are: error detection and re-submission, audit trails, discrepancy reports, error statistics, and backup and recovery. Error detection and re-submission, and audit trails controls are discussed below.

- Error Detection and Re-submission
  Until now we have talked about data control techniques designed to screen the incoming data in order to reject any transactions that do not appear valid, reasonable, complete, etc. Once these errors have been detected, we need to establish specific control techniques to ensure that all corrections are made to the transactions in error and that these corrected transactions are reentered into the system. Such control techniques should include:

  - Having the control group enter all data rejected from the processing cycle in an error log by marking off corrections in this log when these transactions are reentered; open items should be investigated periodically.

  - Preparing an error input record or report explaining the reason for each rejected item. This error report should be returned to the source department for correction and re-submission. This means that the personnel in the originating or source department should have instructions on the handling of any errors that might occur.

  - Submitting the corrected transactions through the same error detection and input validation process as the original transaction.

- Audit Trails Controls
  Another important aspect of the processing cycle is the audit trail. The audit trail consists of documents, journals, ledgers, and worksheets that enable an interested party (e.g., the auditor) to trail an original transaction forward to a summarized total or from a summarized total backward to the original transaction. Only in this way can they determine whether the summary accurately reflects the business's transactions.

9.1.7.2.7 Cost versus Benefit of Controls

In information systems there is a cost associated with each control. The cost of these controls needs to be evaluated as no control should cost more than the potential errors it is established to detect, prevent, or correct. Also, if controls are poorly designed or excessive, they become burdensome and may not be used. The failure to use controls is a key element leading to major risk exposures.

Preventive controls are generally the lowest in cost. Detective controls usually require some moderate operating expense. On the other hand, corrective controls are almost always quite

expensive. As noted above, prior to installing any control, a cost/benefit analysis should be made.

Controls need to be reviewed continually. This is a prime function of the auditor. The auditor should determine if controls are effective. As the result of such a review, an auditor may recommend adding, eliminating, or modifying system controls.

9.2 Internal Control Models

There are three generally accepted models for risk and internal control. These are the COSO Enterprise Risk Management Model, the COSO Internal Control Model, and the CobiT Model.

9.2.1 COSO Enterprise Risk Management (ERM) Model

9.2.1.1 The ERM Process

In Fall 2001, the Committee of Sponsoring Organizations of the Treadway Commission (COSO) launched a landmark study designed to provide guidance in helping organizations manage risk. Despite an abundance of literature on the subject, COSO concluded there was a need for this study to design and build a framework and application guidance. PricewaterhouseCoopers was engaged to lead this project.

The framework defines risk and enterprise risk management, and provides a foundational definition, conceptualizations, objectives categories, components, principles and other elements of a comprehensive risk management framework. It provides direction for companies and other organizations in determining how to enhance their risk management architectures, providing context for and facilitating application in the real world. This document is also designed to provide criteria for companies' use in determining whether their enterprise risk management is effective and, if not, what is needed to make it so.

9.2.1.2 Components of ERM

ERM consists of eight interrelated components. These are derived from the way management runs a business, and are integrated with the management process. The following components are illustrated in Figure 9-2:

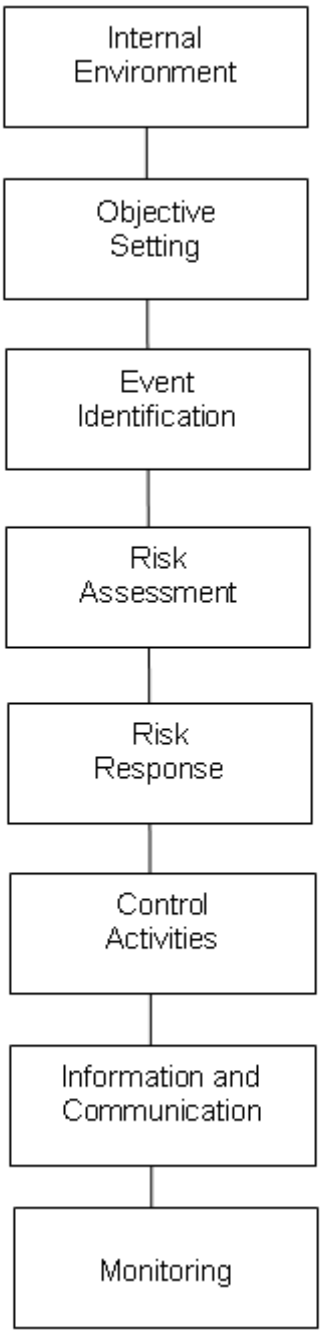


Figure 9-2 The Eight ERM Components

- Internal Environment

  Management sets a philosophy regarding risk and establishes a risk appetite. The internal environment sets the foundation for how risk and control are viewed and addressed by an organization's people.

- Objective Setting

  Objectives must exist before management can identify events potentially affecting their achievement. ERM ensures that management has a process in place to set objectives and that the chosen objectives support and align with the organization's mission/vision and are consistent with the organization's risk appetite.

- Event Identification

  Potential events that might have an impact on the organization must be identified. Event identification includes identifying factors – internal and external – that influence how potential events may affect strategy implementation and achievement of objectives.

- Risk Assessment

  Identified risks are analyzed in order to form a basis for determining how they should be managed. Risks are associated with related objectives that may be affected.

- Risk Response

  Management selects an approach or set of actions to align assessed risks with the organization's risk appetite, in the context of the strategy and objectives.

- Control Activities

  Policies and procedures are established and executed to help ensure that the risk responses management selected are effectively carried out.

- Information and Communication

  Relevant information is identified, captured and communicated in a form and timeframe that enable people to carry out their responsibilities.

- Monitoring

  The entire enterprise risk management process must be monitored, and modifications made as necessary.

9.2.2 COSO Internal Control Framework Model

In the COSO internal control framework, those developing the framework chose to use "control objectives" as opposed to defining risk. However, it is important to recognize that in accomplishing the control objectives, the control designers may have to go through a risk assessment process.

In understanding and using COSO to evaluate internal control, the internal auditor must evaluate whether controls are adequate to achieve the defined control objectives. Throughout the internal control framework only control objective will be defined. Even in the category that COSO defines as "risk", positive control objectives will be stated rather than defining risks.

COSO uses the term “framework” to indicate an integrated system of internal controls. While the COSO framework defines specific control objectives, the framework also indicates that these control objectives are integrated vertically and horizontally.

Internal auditors are normally involved in auditing what COSO refers to as “control activity.” For example, payroll is a control activity. Within the payroll system there are procedures, which produce paychecks and the appropriate records, associated with payroll. In conjunction with this process is the system of controls that commences as transactions are initiated and concludes when those transactions are completed and incorporated into the appropriate organizational financial records. Thus, there are single controls and systems of controls.

COSO’s internal control framework consists of five interrelated components. These are derived from the way management runs a business, and are integrated with the management process. The components are:

- Control Environment – The core of any business is its people – their individual attributes, including integrity, ethical values and competence – and the environment in which they operate. They are the engine that drives the organization and the foundation on which everything rests.
- Risk Assessment – The organization must be aware of, and deal with, the risks it faces. It must set objectives, integrated with the sales, production, marketing, financial and other activities so that the organization is operating in concert. It also must establish mechanisms to identify, analyze and manage the related risks.
- Control Activities – Control policies and procedures must be established and executed to help ensure that the actions identified by management as necessary to address risks to the organization’s objectives are effectively carried out. The control activities component controls transaction processing.
- Information and Communication – Surrounding these activities are information and communication systems. These enable the organization’s people to capture and exchange the information needed to conduct, manage and control its operations.
- Monitoring – The entire process must be monitored, and modifications made as necessary. In this way, the system can react dynamically, changing as conditions warrant.

These internal control components and their linkages are depicted in a model, presented in Figure 9-3. The model depicts the dynamics of internal control systems. Internal control is not a serial process, where one component affects only the next. It is a multidirectional interactive process in which almost any component can and will influence another.

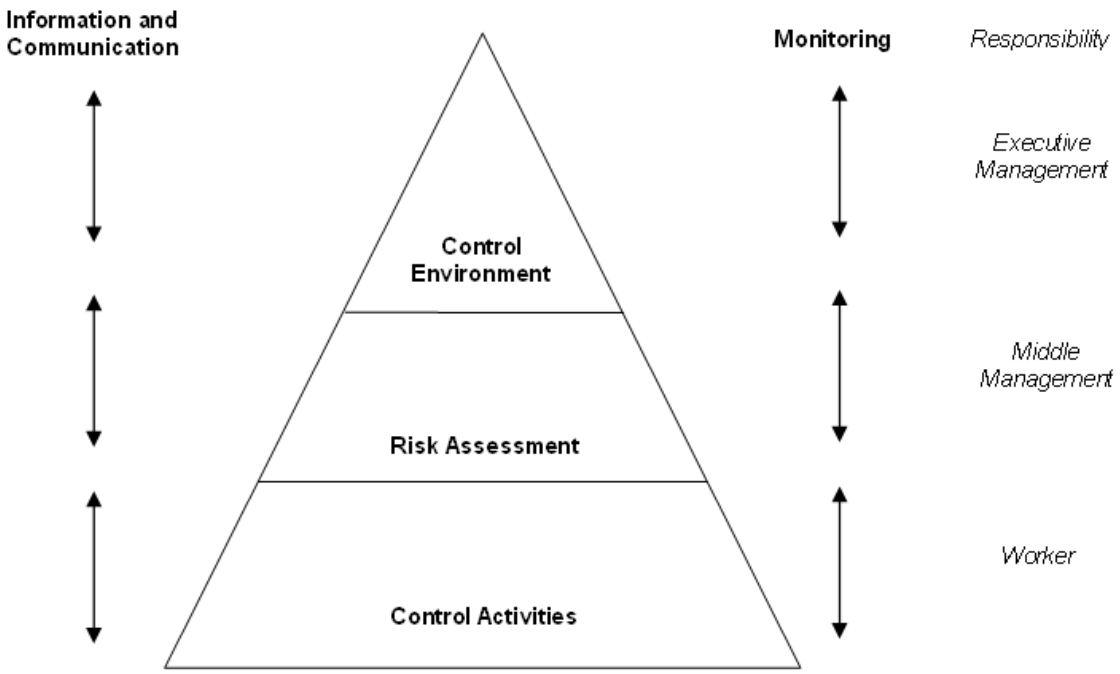


Figure 9-3 COSO Internal Control Framework Components

No two entities will, or should, have the same internal control system. Companies and their internal control needs differ dramatically by industry, size, culture, and management philosophy. Thus, while all entities need each of the components to maintain control over their activities, one company's internal control system often will look very different from another's.

9.2.2.1 Example of a Transaction Processing Internal Control System

Control objectives are defined to minimize risk. Many controls may be implemented to achieve a control objective. All the controls used to accomplish a control objective must be viewed as integrated, that is, a system of internal controls.

In understanding the COSO framework, and the interrelationship of control objectives, you may find it helpful to visualize the framework as a cause/effect diagram like the one shown in Figure 9-4. In viewing this cause/effect diagram from an internal control perspective the effect is the achievement of a control objective. A previously discussed control example was "All products shipped are invoiced". This is the effect that is wanted. Figure 9-4 lists four causes, which if effectively implemented, should achieve the control objective. These causes are that pre-numbered invoices will be used, pre-numbered shipping documents will be used, invoices will be prepared prior to shipment and invoices will be matched to shipping documents.

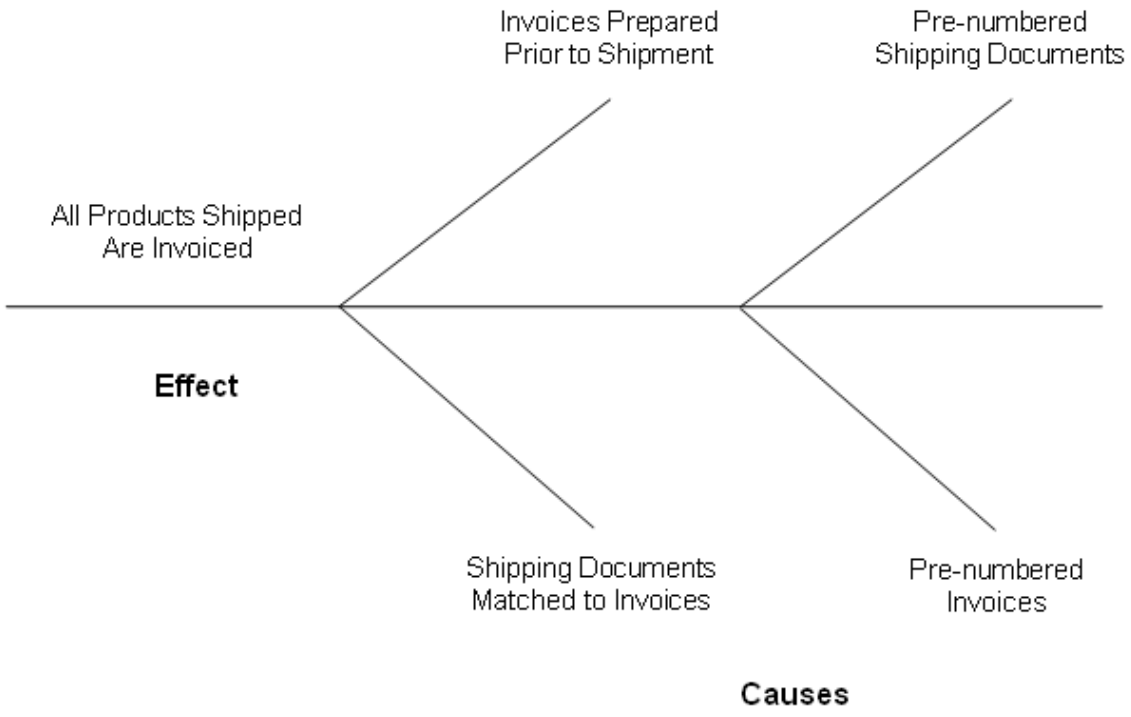


Figure 9-4 Cause and Effect Diagram Example

In understanding the COSO framework, and the interrelationship of control objectives, you may find it helpful to visualize the framework as a cause/effect diagram. The desired product of the COSO framework is the accomplishment of these three control objectives:

- Effective and efficient use of organization's resources
- Preparation of reliable public financial statements
- Organization's compliance to applicable laws and regulations

Using the COSO internal control framework to evaluate internal control is a two-step process as follows:

1. Evaluate the organization's system of controls to assure that each control objective is achieved

2. Assure that for all five components there is an effective integration into the organization's system of controls

9.2.3 CobiT Model

The CobiT model is a generally applicable and accepted standard for IT security and control practices that provides a reference framework for management, users, and IT audit, control and

security practitioners. CobiT enables an enterprise to implement effective governance over IT that is pervasive and intrinsic throughout the enterprise.

The CobiT Model is comprised of the following four-part cycle. The components of the four parts of the CobiT cycle can best be explained by listing the tasks within each component as follows:

- Part 1: Plan and Organize

  The tasks in this part include: Define strategic IT plan.

- Part 2: Acquire and Implement

  The tasks in this part include: Identify automated solutions.

- Part 3 – Deliver and Support

  The tasks in this part include: Defining and managing service levels, performance, problems and incidences.

- Part 4 – Monitor

  The tasks in this part include: Managing the processes and internal control practices.

9.3 Testing Internal Controls

Internal control models emphasize the importance of environmental controls. However, these controls are specified by management and assessed by auditors. Software testers need to focus on testing to determine whether or not the control requirements are effectively implemented. However, testers still have the responsibility to challenge incomplete, incorrect control requirements.

There is a significant difference between auditors and testers. Auditors have the responsibility to assess the adequacy of internal controls. To do this they need to examine all five components of the COSO internal control model. Software testers do not assess adequacy, software testers test to assure control requirements are testable, and then test to determine whether or not the controls were implemented as specified.

Testing the controls in a software system involves accomplishing these objectives:

- The requirements for the controls have been defined. These are normally the risks that need to be minimized or eliminated.
- The defined controls are in place and working, which is traditional testing.
- Test that the “enterprise” controls are included in the software system and are working. Enterprise controls are those controls specified by the enterprise for all software systems. Examples of enterprise controls include security controls and control documentation.

The best practices for testing the controls in a software system involve two tasks:

- Perform Risk Assessment

  Test the completeness of the control requirements, which involves evaluating the software system risks.

- Test Transaction Processing Controls

  Follow the flow of transaction processing to test whether or not the defined controls are in place, working, and that the transactions are controlled throughout transaction processing.

9.3.1 Perform Risk Assessment

Building controls starts with risk assessment because reduction in risk is the requirement for a control. Risk assessment allows an organization to consider the extent to which potential events might have an impact on achievement of objectives. Management should assess events from two perspectives; first, the likelihood of an event occurring and second, the impact of that event. The assessment normally uses a combination of qualitative and quantitative methods.

The positive and negative impacts of potential events should be examined, individually or by category, across the organization. Potentially negative events are assessed on both an inherent and residual basis.

In risk assessment, management considers the mix of potential future events relevant to the organization and its activities. This entails examining factors including organization size, complexity of operations and degree of regulation over its activities that shape the organization's risk profile and influence the methodology it uses to assess risks.

The risk assessment component of Enterprise Risk Management or ERM is comprised of these sub-components:

- Inherent and Residual Risk

  Management considers both inherent and residual risk. Inherent risk is the risk to an organization in the absence of any actions management might take to alter either the risk's likelihood or impact. Residual risk is the risk that remains after management responds to the risk.

- Estimating Likelihood and Impact

  Likelihood represents the possibility that a given event will occur, while impact represents its affect. Estimating the likelihood and impact will determine how much attention the organization should give to a specific event.

- Qualitative and Quantitative Methodology and Techniques

  Qualitative techniques such as categorizing an event into high, medium and low are used where risks do not lend themselves to quantification or when sufficient data is not available for quantification. Quantitative assessment techniques usually require a higher degree of effort and rigor.

- Correlation of Events

  Management may assess how events correlate, where sequences of events combine and interact to create significantly different probabilities or impacts. While the impact of a single event might be slight, a sequence of events might have more significant impact.

Risk assessment is important because it is the process that enables management to determine both the likelihood and potential impact from the materialization of risk. Until the potential impact of an event is known, management may provide too much attention to an event or not enough attention.

Let's look at an example of the risk of customers leaving a Web site because it is too difficult to navigate. If few customers leave because of navigation problems and their purchase potential is minimal, no navigation improvements are needed. On the other hand, if there is a likelihood that many customers will leave with a potentially large loss of orders, resources should be allocated for navigation improvement.

Note that if the software development team performed this task, they can provide the testers with control requirements. If they did not, the testers should work with the users and developers to define the control requirements for test purposes.

9.3.2 Test Transaction Processing Controls

System controls for computer applications involve automated and manual procedures. Automated procedures may include data entry performed in user areas, as well as the control of the data flow within a computer system. Manual procedures in user areas are developed to ensure that the transactions processed by IT are correctly prepared, authorized, and submitted to IT.

Manual application control procedures are also required within IT. For example, the IT input/output control section frequently balances and reconciles input to output. File retention and security procedures may be required and specified for individual computer applications. Such controls are unique to the requirements of the application and complement management controls that govern input/output controls and the media library.

Figure 9-5 shows the six steps of a transaction flow through a computer application system. Transaction flow is used as a basis for classifying transaction processing controls, because it provides a framework for testing the controls for transaction processing. Refer to "Principles and Concepts of Internal Control" for a discussion of the types of controls testers should expect to find in software systems.

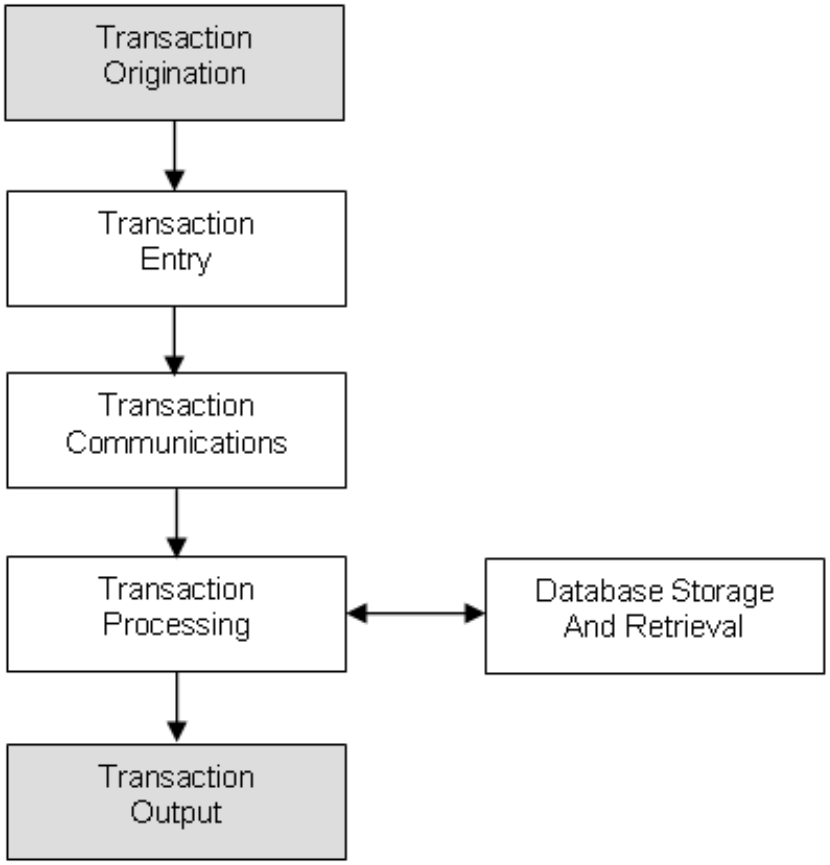


Figure 9-5 Model for Testing Transaction Processing Controls

The two shaded boxes on the figure involve mostly the user organization. Each box is described below.

9.3.2.1 Transaction Origination

Transaction origination controls govern the origination, approval, and processing of source documents and the preparation of data processing input transactions and associated error detection and correction procedures.

9.3.2.2 Transaction Entry

Transaction entry controls govern the data entry via remote terminal or batch, data validation, transaction or batch proofing and balancing, error identification and reporting, and error correction and reentry.

9.3.2.3 Transaction Communications

Transaction communication controls govern the accuracy and completeness of data communications, including message accountability, data protection hardware and software, security and privacy, and error identification and reporting.

9.3.2.4 Transaction Processing

Transaction processing controls govern the accuracy and completeness of transaction processing, including the appropriateness of machine-generated transactions, validation against master files, and error identification and reporting.

9.3.2.5 Database Storage and Retrieval

Transaction database storage and retrieval controls govern the accuracy and completeness of database storage, data security and privacy, error handling, backup, recovery, and retention.

9.3.2.6 Transaction Output

Transaction output controls govern the manual balancing and reconciling of input and output (within the input/output control section and at user locations), distribution of data processing output, control over negotiable documents (within data processing and user areas), and output data retention.

As a general rule, if risks are significant, controls should be strong. If the quality assurance analysts and/or the individual developing the adequacy opinion can match the risks with controls, the opinion can be based on that documentation.

9.4 Testing Security Controls

Testers are not security experts. However, security is too important to organizations for testers to ignore. The following test tasks can add value to the tester's activities:

1. Understand the points where security is most frequently penetrated; and understand the difference between accidental and intentional loss.

2. Build a penetration point matrix to identify software system vulnerabilities; and then investigate the adequacy of the security controls at the point of greatest potential penetration.

3. Assess the security awareness training program to assure the stakeholders in security are aware of their security responsibilities.

4. Understand the attributes of an effective security control.
5. Understand the process for selecting techniques to test security.

9.4.1 Task 1 –Where Security is Vulnerable to Penetration

Data and report preparation areas and computer operations facilities with the highest concentration of manual functions are areas most vulnerable to having security penetrated. Nine primary IT locations are listed, described, and ranked according to vulnerability in Figure 9-6

.

	Vulnerable Areas	**Rank**
IT Locations	Data and report preparation facilities	1
	Computer operations	2
	Non-IT areas	3
	Online storage	4
	Programming offices	5
	Online data and report preparation	6
	Digital media storage facilities	7
	Online operations	8
	Central processors	9

Figure 9-6 Vulnerability to Computer Security According to IT Location

1. Data and Report Preparation Facilities

   Vulnerable areas include key, computer job setup, output control and distribution, data collection, and data transportation. Input and output areas associated with remote terminals are excluded here.

2. Computer Operations

   All locations with computers in the immediate vicinity and rooms housing central computer systems are included in this category. Detached areas that contain peripheral equipment connected to computers by cable and computer hardware maintenance areas or offices are also included. Online remote terminals (connected by telephone circuits to computers) are excluded here.

3. Non-IT Areas

   Security risks also derive from business decisions in such non-IT areas as management, marketing, sales, and business offices; and primary abusive acts may originate from these areas.

4. Online Systems

   The vulnerable functional areas are within online systems, where acts occur by execution of programmed instructions as generated by terminal commands.

5. Programming Offices

   This area includes office areas in which programmers produce and store program listings and documentation.

6. Online Data and Report Preparation

   This category includes the functions for preparing online scripts.

7. Digital Media Storage Facilities

   This area includes data libraries and any storage place containing usable data.

8. Online Operations

   This category is the equivalent of the computer operations discussed previously, but involves the online terminal areas.

9. Central Processors

   These IT areas are within computer systems themselves, and abusive acts may originate from within the computer operating system (not from terminals).

9.4.1.1 Accidental versus Intentional Losses

Errors generated during labor-intensive detailed work lead to vulnerabilities. The errors are usually data errors, computer program errors (bugs), and damage to equipment or supplies. Such errors often require running of jobs multiple times, error correction, and replacement or repair of equipment and supplies.

An *accidental loss* is when someone makes a mistake; an intentional loss is when that mistake is intentional. It is often difficult to distinguish between accidental loss and intentional loss. In fact, some reported intentional loss is due to perpetrators discovering and making use of errors that result in their favor. When loss occurs, employees and managers tend to blame the computer hardware first, in order to absolve themselves from blame and to pass the problem along to the vendor to solve. The problem is rarely a hardware error, but proof of this is usually required before searching elsewhere for the cause. The next most common area of suspicion is users or the source of data generation because, again, the IT department can blame another organization. Blame is usually next placed on the computer programming staff. Finally, when all other targets of blame have been exonerated, IT employees suspect their own work.

It is not uncommon to see informal meetings between computer operators, programmers, maintenance engineers, and users arguing over who should start looking for the cause of a

loss. The thought that the loss was intentional is remote because they generally assume they function in a benign environment.

In many computer centers, employees do not understand the significant difference between accidental loss from errors and intentionally caused losses. Organizations using computers have been fighting accidental loss for 40 years, since the beginning of automated data processing. Solutions are well known and usually well applied relative to the degree of motivation and cost-effectiveness of controls. They anticipate, however, that the same controls used in similar ways also have an effect on people engaged in intentional acts that result in losses. They frequently fail to understand that they are dealing with an intelligent enemy who is using every skill, experience, and access capability to solve the problem or reach a goal. This presents a different kind of vulnerability, one that is much more challenging and that requires adequate safeguards and controls not yet fully developed or realized, let alone adequately applied.

9.4.2 Task 2 – Building a Penetration Point Matrix

There is a dilemma in the question where to test security. Security is needed to protect the resources of the organization. People are the security problem and therefore security should be placed over people. Watching people is not a practical or desirable way to control people.

Computer security is best achieved through controlling activities. The activities in turn control people. For example, we want to stop people from removing computer media from the media library unless they are so authorized. This can best be accomplished by placing controls over the computer media in the form of a librarian; we can then exercise our security procedures through the computer media library and librarian.

This task identifies the activities that need control, as well as the data flow points where penetration is most likely to occur. These concepts are used to build a penetration point matrix that helps identify computer security vulnerabilities for management action. The creation of the penetration point matrix answers the question as to where security is needed and whether or not security controls exist at the most likely point of penetration.

9.4.2.1 Controlling People by Controlling Activities

The computer security challenge in any organization is to control people – not only employees but vendors, customers, passers-by, and ex-employees. The only effective way to control people is to continually monitor their activities. To control an individual we would have to hire another individual to watch him first, and then hire a third individual to watch the second individual watching the first individual. Not only is this not practical, but it would be resented by a large percentage of employees and customers. Thus, another approach is needed to accomplish the same objective.

The solution to the computer security dilemma is to establish activities that control people. For example, it would be difficult to keep unauthorized individuals out of the computer center,

unless strategies like card-access control were initiated. Control designers refer to this concept as division of responsibilities. The activities that appropriately divide responsibilities also introduce the controls that monitor people activities on a continuous basis. The monitoring through activities is not considered objectionable, though constant surveillance would be.

The following challenges exist to building an effective security system and determining the scope or magnitude of the program:

- The identification of the appropriate activities
- The number and types of activities selected for control

9.4.2.2 Selecting Computer Security Activities

The activities requiring security controls can be divided into the following three areas:

- Interface to the computer environment

  These are the activities and individuals who utilize the computer resources in the performance of their work. The specific activities relate to functions either needed by the computer environment or furnished to the computer environment.

- Development activities

  These are the activities relating to the acquisition, creation, and maintenance of the software needed to accomplish the processing requirements established by users of computer facilities to meet business needs.

- Computer operations

  These are the procedures and methods used to process data on the computer using the software developed for that purpose as initiated by the activities that interface to the computer center. Activities also include supporting tasks necessary to ensure the integrity of the mainline operations.

The specific activities included within these three areas are listed in Table 9-1and then discussed in more detail.

Table 9-1: Activities Requiring Security Controls

Interface Activities	Development Activities	Operations Activities
Users of application data and programs	Policies, procedures, and standards	Computer processing
Technical interface to the computer environment	Training	Media libraries
Development and maintenance of application systems	Database administration	Error handling
Privileged users	Communications	Production library control
Vendor interfaces	Documentation	Computer operations
	Program Change Control	Disaster planning
	Records retention program	Privileged utilities and commands

Organizations may wish to divide their activities into different categories, or add other categories of activities subject to control. It is generally advisable to select activities that closely relate to the organizational structure of the company. For example, if the records retention program and media library are under the same individual, it would not be necessary to break these into two distinct activities. On the other hand, the policies, procedures, and standards activities may involve several organizational units and therefore should be broken into two or more different activities.

9.4.2.2.1 Interface Activities

- Users of application data and programs

  Users are the operational activities for which the applications have been developed and for which the processing results are needed. The primary users of computer resources are the operational areas responsible for the application being processed. Secondary users include various staff units in the organization.

- Technical interface to the computer environment

  The computer environment includes many system software packages, for example, operating systems, database management systems and administrative scheduling systems. These individual packages need to be generated and installed; then the interfaces between the packages need to be established. Many of the technical interfaces are performed by systems programmers and other specialists such as database administrators.

- Development and maintenance of application systems

  Application systems are the software packages that process user data to produce the results needed by the users. These application systems can be developed from internally generated specifications, acquired as commercially available software, or developed under contract to vendors who develop applications on a fee basis. The activity includes testing to ensure that the application functions correctly, and then

making any change necessary to ensure the operational currentness of the application. These applications can be developed by the professional data processing staff or by the users themselves.

- Privileged users

  Each organization has a group of users who by their stature in the organization are privileged. This means that they may not be subject to the same level of control as non-privileged users. The two primary categories of privileged users are senior management and auditors. Other privileged users may be specialists within the data processing area or senior data processing management.

- Vendor interfaces

  Organizations contract with a variety of vendors for special services. These include the vendors of hardware, software, and other support services such as contract maintenance, contract cleaning, and contract consulting services. In the performance of vendors' duties, it may be necessary for vendor personnel to interact with computer operations during normal operating periods.

9.4.2.2.2 Development Activities

- Policies, procedures, and standards

  The data processing organization develops policies on how the function is to be performed. These policies are implemented through procedures, such as system development methods by which data processing work is performed. These standards can apply to both the professional data processing area and other users of data processing resources, such as microcomputer users.

- Training

  Training is one of the key attributes of a quality data processing organization. Dr. W. Edwards Deming, the individual given credit for the turnaround of the Japanese economy after the Second World War, states that training is one of the keys to quality data processing. Dr. Deming's philosophy states that individuals should be fully trained in how to perform their job and then evaluated by supervision to ensure that they have mastered those skills. Once fully trained, the individual can then operate with minimal supervision and be expected to produce high-quality work.

- Database administration

  Databases are groupings of data that are managed independently of the application programs that utilize the data. The creation of the databases requires a new organization structure to manage and administer the use of this new development. In many organizations, the database also includes the definition of data and the use of the data dictionary software documentation tool.

- Communications

  This activity encompasses the electronic movement of data between one computer facility and another. In most organizations, the communication facilities involve the use of common carrier lines. When common carrier facilities are used, the

organization loses control over the security of information from the time it passes into the hands of the common carrier until it is again returned to the organization.

- Documentation

  Documentation includes all of the narrative information developed and maintained about data processing activities. In the developmental application, it involves record definitions, system specifications, program listings, test conditions and results, operator manuals, user manuals, control documentation, flow charts, and other pictorial representations. Note that the documentation may be in hard copy format, or may be maintained on electronic media.

- Program change control

  The maintenance activity has the responsibility to define, implement and test changes to application systems. Nevertheless, the control of those changes should be independent of the activity that actually performs the program maintenance. The program change control activity involves logging changes, monitoring their implementation, and verifying that all of the changes to programs are appropriately authorized and that all authorized changes are made.

- Records retention program

  This activity is designed both to retain needed computer-related documents and to appropriately destroy unneeded computer documents. While the computer media is designed to physically store the data, the records retention program relates to the amount of time that the information will be retained. The records retention program includes both manual and computer media. The time and method by which data will be destroyed is an important part of the records retention program. Many organizations either shred or burn key hard-copy computer documentation. In addition, some organizations have custodians to retain and control important records.

9.4.2.2.3 Operations Activities

- Computer processing

  This is the activity of processing data to produce desired results. Processing is used in this context to indicate the totality of steps performed between the initiation of a transaction and the final termination of that transaction. Processing includes both manual and automated functions that manipulate data.

- Media libraries

  Media libraries are repositories for computer media. The most common media are disks, and diskettes. The media libraries may be on-site and off-site. Off-site libraries are used to protect data in the event of a disaster to the on-site media library.

- Error handling

  This activity begins when data is rejected from normal processing and continues until the time the problem has been resolved and the transaction has been correctly processed. Error handling normally involves a logging of errors and then a monitoring

of the correction and reentry process. It is a particularly vulnerable point in many application systems because the reentry may only be subject to minimal control.

- Production library control

  The production library is the repository for computer programs and program-related parameters. For example, job control language statements are necessary to support programs, but are retained in libraries other than the production library. There are many libraries, but the emphasis in this activity is on control over those libraries that affect the integrity of computer processing.

- Computer operations

  These are the steps involved in ensuring that the desired results are achieved through computer processing. Operations involve terminal usage, support operations such as off-line printers and office systems, and the central computer facility. Operations can also occur at off-site service centers.

- Disaster planning

  Disaster planning encompasses the retention of data for purposes other than normal operations, and all of the procedures and methods needed to restore the integrity of operation in the event that it is lost. Since disasters can occur at any point of activity – for example, at a terminal operation – there may be many different activities included within the disaster plan. It is generally advisable to involve users in the development of the plan affecting their operations.

- Privileged utilities and commands

  Various aids are employed to assist the technicians, operators, and developers in the performance of their job responsibilities. Many of these utilities and aids are designed to circumvent the normal operation controls in order to resolve a problem.

9.4.2.3 Controlling Business Transactions

The second dimension of the security program concerns controlling application processing. The activities are designed to support transaction processing. The primary objective of the data processing function is to process data, or process business transactions.

The security of transaction processing occurs at those points where there is transaction activity. Any time a transaction is originated, moved, stored, retrieved, or processed it is subject to unauthorized and unintentional manipulation.

When developing security over transaction processing, it is important to identify the point where the transaction could be manipulated. These points are where the risk of manipulation is greatest and thus where control should be established. Most organizations refer to these points as control points. These ten control points represent the points at which transactions are vulnerable to penetration. If security is to be broken, it will be broken at one of these points. Invariably, the system will be penetrated at the weakest point.

The ten control points in transaction processing are as follows (note that this is an expanded list for security testing from the control model in Figure 9-6):

- Transaction origination

  The creation of a business transaction through the normal conduct of business activities is the first opportunity for manipulation. An order received from a customer would originate from the business transaction of filling a customer order.

- Transaction authorization

  It is management's responsibility to ensure that transactions are only processed in accordance with the intent of management. The method by which this is achieved is to require transactions to be authorized prior to processing. In some instances, this requires a special authorization, such as signing a purchase order; in other instances, management has authorized a transaction if it meets predetermined criteria, such as an order from a customer whose credit has been approved by management.

- Data entry

  This process transcribes transactions onto computer media. In some instances, the transaction is both originated and authorized at the point where it is entered. Nevertheless, these are still distinct control events to be performed.

- Transaction communication

  This control point relates to all the movement activities of transactions. Although shown as a single control point, it may be repeated several times during the life of a single transaction. For example, a transaction can be moved between the origination and authorization step, as well as from the output control point to the usage control point.

- Transaction storage

  This point involves placing transactions in a location to wait further processing. Like communication, it is shown as a single control point but may occur several times in the transaction life cycle. For example, the paper document that originates the transaction may be stored in a file cabinet, whereas the electronic image of that transaction is stored in a database.

- Transaction processing

  Processing encompasses all mathematical and logical manipulations on data, as well as the updating of information stored in computer files. Processing can be automated (by computer) or manual (by people).

- Transaction retrieval

  This control point involves the removal of transactions from storage. As with the storage function, this can be a manual storage cabinet or a computerized file. The removal can be the electronic image; in this case, the image is retained on file. There is also a form of retrieval in which no document or image is retained after the retrieval action is concluded.

- Transaction preparation (output)

This action involves the conversion of electronic media to a format usable by people. It may involve the display of a single transaction on a terminal, or it may involve the consolidation, summarization, and presentation of large volumes of transactions on reports. The content may be altered in this process; for example, state codes may be converted to the formal spelling of the state name.

- Transaction usage

  This involves actions taken on computer-produced results by either people or programs to meet user needs. Actions can range from doing nothing, which in many instances is a definitive action, to initiating a whole series of steps, for example, reordering products.

- Transaction destruction

  This final action on a transaction is the destruction of the transaction itself. In many instances, organization policy and/or the law specifies the time that a transaction must be retained before it can be destroyed. In other instances, the destruction of a transaction is up to the individual responsible for the transaction processing.

There is a close relationship between the processing activities and the control points in transaction processing. The transactions are processed by the previously described activities. These activities either directly contribute to the processing (for example, the communication activity) or support the processes that carry out the transaction (for example, the program change control activity ensures that the programs that perform the processing are current with business requirements).

9.4.2.4 Characteristics of Security Penetration

Many hundreds of years ago, the Chinese built a great wall around their entire civilization to protect themselves from penetrators. This was a costly and time-consuming exercise, and in the end proved futile. The French tried the same tactic by building the Maginot line after World War I, only to find that the Germans went around these great fortifications, which in the end proved to be a useless defense.

A smarter strategy is to locate security defenses at the point where penetration could be the greatest. To select those points, we need to analyze the history of penetrations and develop hypotheses that tell us where our systems are most vulnerable.

We need to explore two premises to understand where penetration will occur. First, penetration will occur at the weakest point in transaction processing. Penetrations aimed at manipulating transaction processing will pick the weakest control point in the processing cycle for penetration. The term hacking means a continual probing to identify a weakness. Penetrators invariably hack until they find the weak point and then penetrate at that point. Therefore, if the weakest point in the cycle is strengthened, the effort required to penetrate the system increases. Each time the weak point is strengthened, the ante to play the penetration game goes up.

Second, penetration will occur in the least-controlled activity. The activity in which there is the greatest opportunity to manipulate is the activity that is most subject to manipulation. For example, if it is easiest to manipulate training, then that is the activity that will be used to penetrate the system. In one of the classic computer fraud cases, the head teller in a bank trained new tellers to ignore the warning messages that indicated unauthorized manipulation was occurring.

The two variables described, control points and controllable activities, hold the key to determining where security is needed. If either a control point or an activity is weak, it needs to be strengthened; and if activities and control points that are related are both weak, the opportunity for penetration is even greater. By looking at these variables and showing the relationship, we can identify the point where the computer processes are most likely to be penetrated.

The Computer Security Penetration-Point Matrix is directed at data manipulation. It is not designed to identify all security threats. For example, natural disasters are a threat, but not a people threat. Disgruntled employees may wish to sabotage the computer center by destroying equipment; this is a threat to computer processing, but not a threat to transaction processing. On the other hand, most of the day-to-day security threats are data related and are identifiable through the use of the Computer Security Penetration-Point Matrix as shown in Table 9-2, “Computer Security Penetration-Point Matrix - Interface Activities”, Table 9-3, “Computer Security Penetration-Point Matrix - Development Activities”, and Table 9-4, “Computer Security Penetration-Point Matrix - Operation Activities”.

These three tables or matrices divide the 19 controllable activities by their management security control groupings. This matrix lists the ten transaction control points in the vertical column and the controllable activities in the horizontal column.

The matrix can be completed for each major business transaction. If the organization has a control design methodology, the insight gained from completing the form for the organization will suffice to identify the major penetration points. If each application is uniquely controlled, the matrix should be prepared for each transaction.

The matrix is completed control point by control point. The processing at each control point is viewed in relation to the activities that are involved in that processing. In most instances, many of the activities will be involved. Several questions need to be asked when looking at each activity. First, is there a high probability that this control point could be penetrated through this activity? For example, the first control point is transaction origination, and the first controllable activity is users of that application. The question then becomes: Do the users have a high probability of penetrating the point where the transaction is originated? If so, three points are allocated and recorded in the intersection of the lines from that control point and controllable activity.

If there is not a high probability of penetration, the question must be asked whether there is an average probability. If so, a score of two is put in the intersection between the control point and controllable activity. If there is a low probability, but still a probability, of penetration, then a score of one should be recorded in the matrix intersection. If there is no probability of penetration, or a minimal probability of penetration, a dash or zero should be put in the

intersection. This procedure is continued until all the control points have been evaluated according to each of the 19 controllable activities.

The scores allocated for each intersection should be totaled vertically and horizontally. This will result in a minimum horizontal score of 0, and a maximum score of 57 (for example, 3 points x 19 = 57). The vertical scores will total a minimum of 0 and a maximum of 30 (for example, 3 points x 10 control points = 30). Circle the high scores in the vertical and horizontal Total columns. These will indicate the high-risk control points and the high-risk activities. Circle the intersections for which there are high scores for the transaction control point, and a high score for the controllable activity and either a two or three in the intersection between those high total scores.

The most probable penetration points should be listed in this order:

- First priority is given to the intersection at which both the controllable activity and the control point represent high probabilities of penetration through high total scores. These are the points where there is the greatest risk of penetration.
- Second priority is given to the high-risk controllable activities. The activities are general controls, which usually represent a greater risk than application control points.
- Third priority is given to the high-risk control points as indicated by high total scores for the control point.

At the end of this security practice, the project leader will have indication of where security is needed most. Because security will be placed on activities and at transaction control points through activities, this identification process is important in determining the magnitude of the computer security program.

Table 9-2: Computer Security Penetration-Point Matrix - Interface Activities

	Computer Security Activities – Interface Activities				
Transaction Control Points	Interface to computer environment	Technical interface to computer environment	Development and maintenance of application systems	Privileged users	Vender interfaces
Transaction origination					
Transaction authorization					
Data entry					
Transaction communication					
Transaction storage					
Transaction processing					
Transaction retrieval					
Transaction preparation (output)					
Transaction usage					
Transaction destruction					

Table 9-3: Computer Security Penetration-Point Matrix - Development Activities

	Computer Security Activities – Development Activities						
Transaction Control Points	Policies, procedures and standards	Training	Database administration	Communications	Documentation	Program change control	Records retention program
Transaction origination							
Transaction authorization							
Data entry							
Transaction communication							
Transaction storage							
Transaction processing							
Transaction retrieval							
Transaction preparation (output)							
Transaction usage							
Transaction destruction							

Table 9-4: Computer Security Penetration-Point Matrix - Operation Activities

	Computer Security Activities – Operation Activities						
Transaction Control Points	Computer processing	Media libraries	Error handling	Production library control	Computer operations	Disaster planning	Privileged utilities and commands
Transaction origination							
Transaction authorization							
Data entry							
Transaction communication							
Transaction storage							
Transaction processing							
Transaction retrieval							
Transaction preparation (output)							
Transaction usage							
Transaction destruction							
TOTAL SCORE							

9.4.3 Task 3 – Assess Security Awareness Training

The best approach for testers to assess the adequacy of their organization's security awareness program is to compare that program against a world-class security awareness program. This section describes a world-class security awareness program. The assessment will identify activities in the world-class security program that are not included in your organization's security awareness program. Based on the results of this assessment, IT management can decide the merits of enhancing their security awareness program.

IT organizations cannot protect the confidentiality, integrity, and availability of information in today's highly networked systems environment without ensuring that all the people involved in using and managing IT:

- Understand their roles and responsibilities related to the organizational mission
- Understand the organization's IT security policy, procedures, and practices
- Have at least adequate knowledge of the various management, operational, and technical controls required and available to protect the IT resources for which they are responsible
- Fulfill their security responsibilities

As cited in audit reports, periodicals, and conference presentations, it is generally agreed by the IT security professional community that people are the weakest link in attempts to secure systems and networks.

The "people factor" – not technology – is the key to providing an adequate and appropriate level of security. If people are the key, but are also a weak link, more and better attention must be paid to this "component of security". A robust and enterprise-wide awareness and training program is paramount to ensure that people understand their IT security responsibilities, organizational policies, and how to properly use them to protect the IT resources entrusted to them.

This practice provides a strategy for building and maintaining a comprehensive awareness and training program, as part of an organization's IT security program. The strategy is presented in a life cycle approach, ranging from designing, developing, and implementing an awareness and training program, through post-implementation evaluation of the program. The security awareness program includes guidance on how IT security professionals can identify awareness and training needs, develop a training plan, and get organizational buy-in for the funding of awareness and training program efforts.

While there is no one best way to develop a security awareness program, the process that follows is an all inclusive process of the best security awareness training program. This example includes these three steps:

1. IT management creates a security awareness policy.

2. Develop the strategy that will be used to implement that policy. (Note that this practice focuses on that strategy. Other practices will explain how to implement the steps in that strategy.)

3. Assign the roles for security and awareness to the appropriate individuals.

9.4.3.1 Step 1 – Create a Security Awareness Policy

The CIO and/or the IT Director need to establish a security awareness policy. The policy needs to state management's intension regarding security awareness. Experience has shown that unless senior management actively supports security awareness, there will be a lack of emphasis on security among the staff involved in using information technology and information.

Management support for security awareness begins with the development and distribution of a security awareness policy. Once that policy has been established, management makes security awareness happen through supporting the development of a strategy and tactics for security awareness, appropriately funding those activities, and then becoming personally involved in ensuring the staff knows of management's support for security awareness.

A security awareness policy can be as simple as ensuring that all stakeholders involved in the use of information technology and the information controlled by that technology, be made aware of their role and responsibility in assuring the security over that technology and information. Generally that policy would be clarified and expanded with statements such as:

"Ensure that all individuals are appropriately trained in how to fulfill their security responsibilities before allowing them access to the system. Such training shall ensure that employees are versed in the rules of the system and apprise them about available technical assistance and technical security products and techniques. Behavior consistent with the rules of the system and periodic refresher training shall be required for continued access to the system. Before allowing individuals access to the application, ensure that all individuals receive specialized training focused on their responsibilities and the application rules. This may be in addition to the training required for access to a system. Such training may vary from a notification at the time of access (e.g., for members of the public usage of an information retrieval application) to formal training (e.g., for an employee that works with a high-risk application)."

"Ensure that the organization has trained personnel sufficient to assist the agency in complying with these requirements and related policies, procedures, standards, and guidelines. Delegate to the agency CIO the authority to ensure compliance with the responsibilities for information security. The required agency-wide information security program shall include security awareness training to inform personnel, including contractors and other users of information systems that support the operations and assets of the agency, or information security risks associated with their activities."

9.4.3.2 Step 2 – Develop a Security Awareness Strategy

A successful IT security program consists of:

1. developing an IT security policy that reflects business needs tempered by known risks
2. informing users of their IT security responsibilities, as documented in the security policy and procedures
3. establishing processes for monitoring and reviewing the program.

Security awareness and training should be focused on the organization's entire user population. Management should set the example for proper IT security behavior within an organization. An awareness program should begin with an effort that can be deployed and implemented in various ways and is aimed at all levels of the organization including senior and executive managers. The effectiveness of this effort will usually determine the effectiveness of the awareness and training program. This is also true for a successful IT security program.

An effective IT security awareness and training program explains proper rules of behavior for the use of an organization's IT systems and information. The program communicates IT security policies and procedures that need to be followed. This must precede and lay the basis for any sanctions imposed due to noncompliance. Users should first be informed of the expectations. Accountability must be derived from a fully informed, well-trained, and aware workforce.

This step describes the relationship between awareness, training, and education. An effective IT security awareness and training program can succeed only if the material used in the program is firmly based on the IT security awareness policy and IT issue-specific policies. If policies are written clearly and concisely, then the awareness and training material – based on the policies – will be built on a firm foundation. Learning is a continuum; it starts with awareness, builds to training, and evolves into education – the awareness-training-education continuum. The continuum shown in Figure 9-7 illustrates the conceptual relationship between awareness, training, and education. For the purpose of this practice, clear boundaries are established between the three methods of learning.

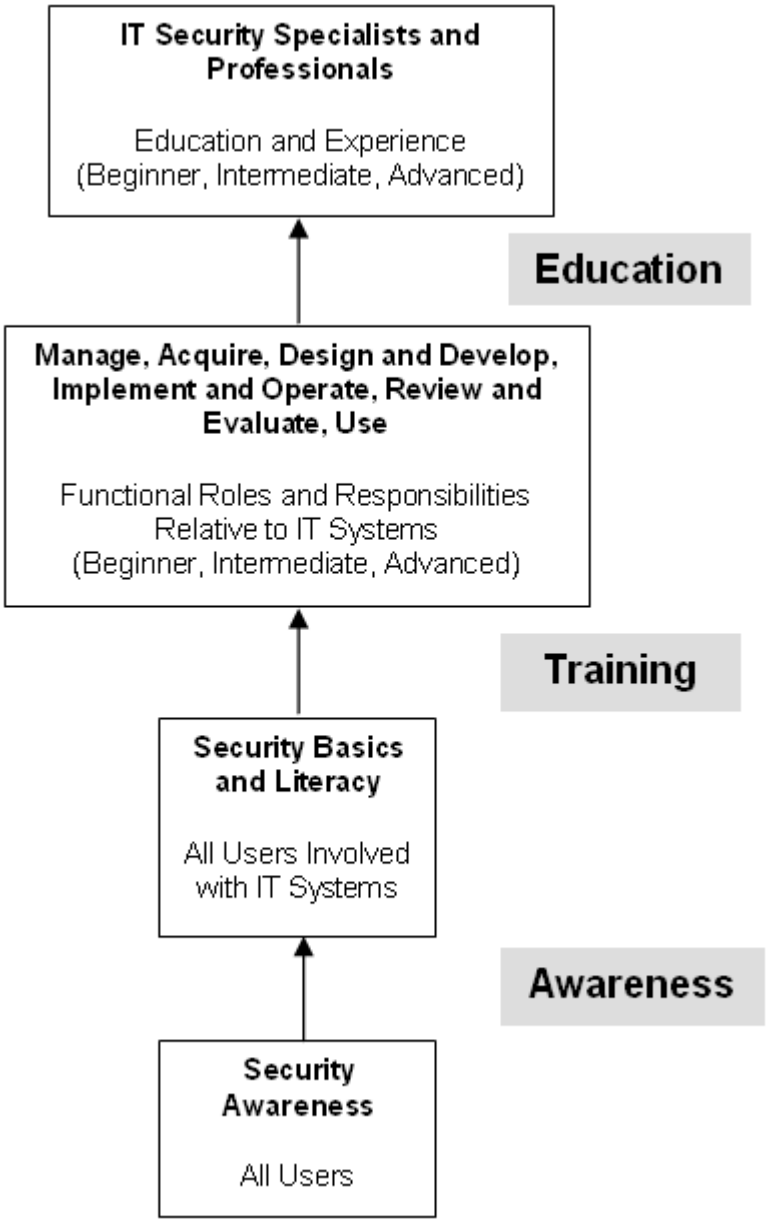


Figure 9-7 The IT Security Learning Continuum

9.4.3.2.1 Awareness

Security awareness efforts are designed to change behavior or reinforce good security practices.

Awareness is not training. The purpose of awareness presentations is simply to focus attention on security. Awareness presentations are intended to allow individuals to recognize IT security concerns and respond accordingly. In awareness activities, the learner is the recipient of information, whereas the learner in a training environment has a more active role. Awareness

relies on reaching broad audiences with attractive packaging techniques. Training is more formal, having a goal of building knowledge and skills to facilitate the job performance.

An example topic for an awareness session (or awareness material to be distributed) is virus protection. The subject can simply and briefly be addressed by describing what a virus is, what can happen if a virus infects a user's system, what the user should do to protect the system, and what the user should do if a virus is discovered.

9.4.3.2.2 Training

Training strives to produce relevant and needed security skills and competencies. Training is defined as follows: The 'training' level of the learning continuum strives to produce relevant and needed security skills and competencies by practitioners of functional specialties other than IT security (e.g., management, systems design and development, acquisition, auditing). The most significant difference between training and awareness is that training seeks to teach skills, which allow a person to perform a specific function, while awareness seeks to focus an individual's attention on an issue or set of issues. The skills acquired during training are built upon the awareness foundation, in particular, upon the security basics and literacy material.

An example of training is an IT security course for system administrators, which should address in detail the management controls, operational controls, and technical controls that should be implemented. Management controls include policy, IT security program management, risk management, and life cycle awareness and training, computer support and operations, and physical and environmental security issues. Technical controls include identification and authentication, logical access controls, audit trails, and cryptography.

9.4.3.2.3 Education

Education is defined as follows: The 'education' level integrates all of the security skills and competencies of the various functional specialties into a common body of knowledge, adds a multi-disciplinary study of concepts, issues, and principles (technological and social), and strives to produce IT security specialists and professionals capable of vision and pro-active response.

An example of education is a degree program at a college or university. Some people take a course or several courses to develop or enhance their skills in a particular discipline. This is training as opposed to education. Many colleges and universities offer certificate programs, wherein a student may take two, six, or eight classes, for example, in a related discipline, and is awarded a certificate upon completion. Often, these certificate programs are conducted as a joint effort between schools and software or hardware vendors. These programs are more characteristic of training than education. Those responsible for security training need to assess both types of programs and decide which one better addresses identified needs.

9.4.3.2.4 Professional Development

Professional development is intended to ensure that users, from beginner to the career security professional, possess a required level of knowledge and competence necessary for their roles. Professional development validates skills through certification. Such development and

successful certification can be termed "professionalization." The preparatory work to test such a certification normally includes study of a prescribed body of knowledge or technical curriculum, and may be supplemented by on-the-job experience.

The movement toward professionalization within the IT security field can be seen among IT security officers, IT security auditors, IT contractors, and system/network administrators, and is evolving. There are two types of certification: general and technical. The general certification focuses on establishing a foundation of knowledge on the many aspects of the IT security profession. The technical certification focuses primarily on the technical security issues related to specific platforms, operating systems, vendor products, etc.

Some organizations focus on IT security professionals with certifications as part of their recruitment efforts. Other organizations offer pay raises and bonuses to retain users with certifications and encourage others in the IT security field to seek certification.

9.4.3.3 Step 3 – Assign the Roles for Security Awareness

While it is important to have a policy that requires the development and implementation of security and training, it is crucial that IT organizations understand who has responsibility for IT security awareness and training. This step identifies and describes those within an organization that have responsibility for IT security awareness and training.

Some organizations have a mature IT security program, while other organizations may be struggling to achieve basic staffing, funding, and support. The form that an awareness and training program takes can vary greatly from organization to organization. This is due, in part, to the maturity of that program. One way to help ensure that a program matures is to develop and document IT security awareness and training responsibilities for those key positions upon which the success of the program depends.

9.4.3.3.1 IT Director/CIO

The IT Director and/or the CIO must ensure that high priority is given to effective security awareness and training for the workforce. This includes implementation of a viable IT security program with a strong awareness and training component. The IT Director should:

- Assign responsibility for IT security
- Ensure that an organization-wide IT security program is implemented, is well-supported by resources and budget, and is effective
- Ensure that the organization has enough sufficiently trained personnel to protect its IT resources
- Establish overall strategy for the IT security awareness and training program
- Ensure that senior managers, system and data owners, and others understand the concepts and strategy of the security awareness and training program, and are informed of the progress of the program's implementation
- Ensure that the IT security awareness and training program is funded
- Ensure the training of personnel with significant security responsibilities

- Ensure that all users are sufficiently trained in their security responsibilities
- Ensure that effective tracking and reporting mechanisms are in place.

9.4.3.3.2 IT Security Program Manager

The IT Security Program Manager has tactical-level responsibility for the awareness and training program. In this role, the program manager should:

- Ensure that awareness and training material developed is appropriate and timely for the intended audiences
- Ensure that awareness and training material is effectively deployed to reach the intended audience
- Ensure that users and managers have an effective way to provide feedback on the awareness and training material and its presentation
- Ensure that awareness and training material is reviewed periodically and updated when necessary
- Assist in establishing a tracking and reporting strategy

9.4.3.3.3 IT Managers

IT Managers have responsibility for complying with IT security awareness and training requirements established for their users. IT Managers should:

- Work with the CIO and IT security program manager to meet shared responsibilities
- Serve in the role of system owner and/or data owner, where applicable
- Consider developing individual development plans (IDPs) for users in roles with significant security responsibilities
- Promote the professional development and certification of the IT security program staff, full-time or part-time security officers, and others with significant security responsibilities
- Ensure that all users (including contractors) of their system (i.e., general support systems and major applications) are appropriately trained in how to fulfill their security responsibilities before allowing them access
- Ensure that users (including contractors) understand specific rules of each system and application
- Work to reduce errors and omissions by users due to lack of awareness and/or training.

9.4.3.3.4 Users

Users are the largest audience in any organization and are the single most important group of people who can help to reduce unintentional errors and IT vulnerabilities. Users may include employees, contractors, foreign or domestic guest researchers, other agency personnel, visitors, guests, and other collaborators or associates requiring access. Users must:

- Understand and comply with the IT security policies and procedures
- Be appropriately trained in the rules of behavior for the systems and applications to which they have access

- Work with management to meet training needs
- Keep software/applications updated with security patches
- Be aware of actions they can take to better protect their information. These actions include, but are not limited to: proper password usage, data backup, proper anti-virus protection, reporting any suspected incidents or violations of security policy, and following rules established to avoid social engineering attacks and rules to deter the spread of spam or viruses and worms.

9.4.4 Task 4 – Understand the Attributes of an Effective Security Control

When testers evaluate a security control they need to understand what makes an effective security control. The following eight security control attributes of an effective security control are designed to help testers determine whether or not a security control is effective.

- Simplicity

  Security mechanisms (and information systems in general) should be as simple as possible. Complexity is at the root of many security issues.

- Fail Safe

  If a failure occurs, the system should fail in a secure manner. That is, if a failure occurs, security should still be enforced. It is better to lose functionality than lose security.

- Complete Mediation

  Rather than providing direct access to information, mediators that enforce access policy should be employed. Common examples include file system permissions, web proxies and mail gateways.

- Open Design

  System security should not depend on the secrecy of the implementation or its components. “Security through obscurity” does not work.

- Separation of Privilege

  Functions, to the degree possible, should be separate and provide as much granularity as possible. The concept can apply to both systems and operators and users. In the case of system operators and users, roles should be as separate as possible. For example, if resources allow, the role of system administrator should be separate from that of the security administrator.

- Psychological Acceptability

  Users should understand the necessity of security. This can be provided through training and education. In addition, the security mechanisms in place should present users with sensible options that will give them the usability they require on a daily basis. If users find the security mechanisms too cumbersome, they find ways to work

around or compromise them. An example of this is using random passwords that are very strong but difficult to remember; users may write them down or look for methods to circumvent the policy.

- Layered Defense

  Organizations should understand that any single security mechanism is generally insufficient. Security mechanisms (defenses) need to be layered so that compromise of a single security mechanism is insufficient to compromise a host or network. There is no “magic bullet” for information system security.

- Compromise Recording

  When systems and networks are compromised, records or logs of that compromise should be created. This information can assist in security of the network and host after the compromise and assist in identifying the methods and exploits used by the attacker. This information can be used to better secure the host or network in the future. In addition, the records and logs can assist organizations in identification and prosecution.

9.4.5 Task 5 – Selecting Techniques to Test Security

Some security testing techniques are predominantly manual, requiring an individual to initiate and conduct the test. Other tests are highly automated and require less human involvement. Regardless of the type of testing, testers that plan and conduct security testing should have significant security and networking knowledge, including expertise in the following areas: network security, firewalls, intrusion detection system, operating systems, programming and networking protocols (such as TCP/IP).

The following security testing techniques are recommended for testing security:

- Network scanning
- Vulnerability scanning
- Password cracking
- Log review
- Integrity checkers
- Virus detection
- War dialing
- War driving (wireless LAN testing)
- Penetration testing

Often, several of these testing techniques are used together to gain more comprehensive assessment of the overall network security posture. For example, penetration testing usually includes network scanning and vulnerability scanning to identify vulnerable hosts and services that may be targeted for later penetration. Some vulnerability scanners incorporate

password cracking. None of these tests by themselves will provide a complete picture of the network or its security posture.

The selection of security testing techniques involves the following three steps which are described below:

- Understand security testing techniques
- Select security testing techniques based on the strengths and weaknesses of those techniques
- Determine the frequency of use of security testing techniques based on the system category

9.4.5.1 Step 1 – Understand Security Testing Techniques

The individual selecting the security testing techniques should be knowledgeable in both security and the available testing techniques for security. Testing information security is a specialized test competency. Table 9-5 on page 53 contains the description of nine frequently used software testing techniques. The description of the technique should be adequate for knowledgeable software security testers but may be inadequate for testers new to the security testing area.

9.4.5.2 Step 2 – Select Security Testing Techniques Based on the Strengths and Weaknesses of Those Techniques

Security testers should have identified a testing objective prior to selecting the security testing techniques. The selection of the technique should be based on the strengths and weaknesses of the technique as it applies to the security test objective. Table 9-5, “Comparisons of Network Testing Techniques” lists the strengths and weaknesses of the nine security testing techniques described in Step 1. The selection of the technique can be accomplished by reviewing those strengths and weaknesses to determine the applicability of that security testing technique to the security testing objective.

Table 9-5: Comparisons of Network Testing Techniques

Type of Technique	Strengths	Weaknesses
Network Scanning	• Fast (as compared to vulnerability scanners or penetration testing) • Efficiently scans hosts, depending on number of hosts in network • Many excellent freeware tools available • Highly automated (for something component) • Low cost	• Does not directly identify known vulnerabilities (although will identify commonly used Trojan ports) • Generally used as a prelude to penetration testing not as final test • Requires significant expertise to interpret results
Vulnerability Scanning	• Can be fairly fast depending on number of hosts scanned • Some freeware tools available • Highly automated (for scanning) • Identifies known vulnerabilities • Often provides advice on mitigating discovered vulnerabilities • High cost (commercial scanners) to low (freeware scanners) • Easy to run on a regular basis	• Has high false positive rate • Generates large amount of traffic aimed at a specific host (which can cause the host to crash or lead to a temporary denial of service) • Not stealthy (e.g., easily detected by IDS, firewall and even end-users [although this may be useful in testing the response of staff and altering mechanisms]) • Can be dangerous in the hands of a novice (especially DOS attacks) • Often misses latest vulnerabilities • Identifies only surface vulnerabilities
Penetration Testing	• Tests network using the methodologies and tools that attackers employ • Verifies vulnerabilities • Goes beyond surface vulnerabilities and demonstrates how these vulnerabilities can be exploited iteratively to gain greater access • Demonstrates that vulnerabilities are not purely theoretical • Can provide the realism and evidence needed to address security issues • Social engineering allows for testing of procedures and the human element network security	• Requires great expertise • Very labor intensive • Slow, target hosts may take hours/ days to “crack” • Due to time required, not all hosts on medium or large sized networks will be tested individually • Dangerous when conducted by inexperienced testers • Certain tools and techniques may be banned or controlled by agency regulations (e.g., network sniffers, password crackers, etc.) • Expensive • Can be organizationally disruptive
Password Cracking	• Quickly identifies weak passwords • Provides clear demonstration of password strength or weakness • Easily implemented • Low cost	• Potential for abuse • Certain organizations restrict use

Table 9-5: Comparisons of Network Testing Techniques (Continued)

Type of Technique	Strengths	Weaknesses
Log Reviews	• Provides excellent information • Only data source that provides historical information	• Cumbersome to manually review • Automated tools not perfect can filter out important information
File Integrity Checkers	• Reliable method of determining whether a host has been compromised • Highly automated • Low cost	• Does not detect any compromise prior to installation • Checksums need to be updated when the system is updated • Checksums need to be protected (e.g., read only (CD-ROM) because they provide no protection if they can be modified by an attacker
Virus Detectors	• Excellent at preventing and removing viruses • Low/Moderate cost	• Require constant updates to be effective • Some false positive issues • Ability to react to new, fast replicating viruses is often limited
War Dialing	• Effective way to identify unauthorized modems	• Legal and regulatory issues especially if using public switched network • Slow
War Driving	• Effective way to identify unauthorized wireless access points	• Possible legal issues if other organization's signals are intercepted • Requires some expertise in computing, wireless networking and radio engineering

9.4.5.3 Step 3 – Determine the Frequency of Use of Security Testing Techniques Based on the System Category

This step proposes two categories of systems subject to security testing. Table 9-6, "Summarized Evaluation and Frequency Factors" describes a general schedule and list of evaluation factors for testing categories. Category 1 systems are those sensitive systems that provide security for the organization that provide other critical functions. These systems often include:

- Firewalls, routers, and perimeter defense systems such as for intrusion detection,
- Public access systems such as web and e-mail servers,
- DNS and directory servers and other internal systems that would likely be intruder targets.

Category 2 systems are generally all other systems, such as those systems that are protected by firewalls, etc., but still must be tested periodically. Use Table 9-6, "Summarized Evaluation

and Frequency Factors" to determine the frequency of security testing based on one of those two system categories.

Table 9-6: Summarized Evaluation and Frequency Factors

Test Technique	Category 1 Frequency	Category 2 Frequency	Benefit
Network scanning	Continuously to Quarterly	Semi-Annually	• Enumerates the network structure and determines the set of active hosts, and associated software • Identifies unauthorized hosts connected to a network • Identifies open ports • Identifies unauthorized services
Vulnerability Scanning	Quarterly or bi-monthly (more often for certain high risk systems), when the vulnerability database is updated	Semi-Annually	• Enumerates the network structure and determines the set of active hosts, and associated software • Identifies a target set of computers to focus vulnerability analysis • Identifies potential vulnerabilities on the target set • Validates that operation systems and major applications are up-to-date with security patches and software versions
Penetration Testing	Annually	Annually	• Determines how vulnerable an organization's network is to penetration and the level of damage that can be incurred • Tests IT staff's response to perceived security incidents and their knowledge of an implementation of the organization's security policy and system's security requirements
Password Cracking	Continuously to same frequency as expiration policy	Same frequency as expiration policy	• Verifies that the policy is effective in producing passwords that are more or less difficult to break • Verifies that users select passwords that are compliant with the organization's security policy
Log Reviews	Daily for critical systems, (e.g., firewalls)	Weekly	• Validates that the system is operating according to policies
Integrity Checkers	Monthly and in case of suspected incidents	Monthly	• Detects unauthorized file modifications
Virus Detectors	Weekly or as required	Weekly or as required	• Detects and deletes viruses before successful installation on the system
War Dialing	Annually	Annually	• Detects unauthorized modems and prevents unauthorized access to a protected network
War Driving	Continuously to weekly	Semi-Annually	• Detects unauthorized wireless access points and prevents unauthorized access to a protected network

This page intentionally left blank.

Skill Category

10

Testing New Technologies

Testers require skills in their organization's current technology, as well as a general understanding of the new information technology that might be acquired by their organization. The new technology skills are required because test plans need to be based on the types of technology used. Also technologies new to the organization and the testers pose technological risks which must be addressed in test planning and test execution. This section addresses the newer IT technologies, but any technology new to the testers or the organization must be addressed in the test plan.

Risks Associated with New Technology	**10-1**
Newer IT Technologies that Impact Software Testing	**10-3**
Testing the Effectiveness of Integrating New Technology	**10-12**

10.1 Risks Associated with New Technology

Testers need to answer the following questions when testing a new project:

- Is new technology utilized on the project being tested?
- If so, what are the concerns and risks associated with using that technology?
- If significant risks exist how will the testing process address those risks?

The following are the more common risks associated with the use of technology new to an IT organization. Note that this list is not meant to be comprehensive but rather representative of the types of risks frequently associated with using new technology:

- Unproven technology

The technology is available but there is not enough experience with the use of that technology to determine whether or not the stated benefits for using that technology can actually be received.

- Technology is defective

  The technology as acquired does not work in the specified manner. For example there could be incompatible processes in building software using agile development, or a flaw in the hardware circuitry.

- Inefficient technology

  The technology fails to achieve the productivity gains associated with that technology.

  Technology incompatible with other implemented technologies

  The technologies currently in place in the IT organization are incompatible with the new technology acquired. Therefore, the new technology may meet all of its stated benefits but the technology cannot be used because of incompatibility with currently implemented technologies.

- New technology obsoletes existing implemented technologies

  Many times when vendors develop new technologies, such as a new version of software, they discontinue support of the existing software version. Thus, the acquisition of new technology involves deleting the existing technologies and replacing it with the new technologies. Sometimes vendors do not declare the current technologies obsolete until there has been general acceptance with the new technology. If testers do not assume that older technologies will become obsolete they fail to address the significant new technology risk.

- Variance between documentation and technology execution

  The manuals and instructions associated with using new technologies may differ from the actual performance of the technologies. Thus when organizations attempt to use the new technologies with the documented procedures the new technologies will fail to perform as specified.

- Staff not competent to use new technology

  Training and deployment processes may be needed to assure the organization has the adequate competency to use the new technology effectively and efficiently. If the organization's staff does not possess the necessary skill sets, they will not gain the benefits attributable to the new technology.

- Lack of understanding how to optimize the new technology

  Studies show that most organizations only use limited aspects of new technology. They do not take the time and effort to learn the technology well enough to optimize the use of the technology. Therefore, while some benefits may be received, the organization may miss some of the major benefits associated with using a new technology.

- Technology not incorporated into the organization's work processes

This is typical implementation of new technologies at technology maturity Level 1. At this level, management cannot control how the new technology will be used in the IT organization. Because staff has the decision over whether or not to use technology and how to use it, some significant benefits associated with that technology may be lost.

- Obsolete testing tools

  The implementation of new technology may obsolete the use of existing testing tools. New technologies may require new testing methods and tools.

- Inadequate vendor support

  The IT staff may need assistance in using and testing the technology, but are unable to attain that assistance from the vendor.

10.2 Newer IT Technologies that Impact Software Testing

The following is a list of newer IT technologies that increase the risk associated with testing software systems. The list is not meant to be exhaustive but rather representative of the types of technologies that many IT organizations consider new that impact software testing.

10.2.1 Web-Based Applications

A Web-based system consists of hardware components, software components, and users. Having an understanding of a Web application's internal components and how those components interface with one another, even if only at a high level, leads to better testing.

Such knowledge allows for the analysis of a program from its developer's perspective – which is invaluable in determining test strategy and identifying the cause of errors. Furthermore, analyzing the relationships between the components leads to an understanding of the interaction of the work product from the perspective of several independent developers, as opposed to from only the individual developer's perspective.

The work product is analyzed from a perspective that is not evident from the analysis of any individual component. You analyze how all these components interact with each other to make up the system. The gray-box tester provides this capability. You look at the system at a level that is different from that of the developer. Just like the black-box tester, you add a different perspective and, therefore, value. Generally, we learn about an application's architecture from its developers during a walkthrough.

An alternate approach is to do your own analysis by tracing communication traffic between components. For example, tests can be developed that hit a database server directly, or on behalf of actual user activities, via browser-submitted transactions. Regardless, we need to

have a firm grasp of typical Web-based application architecture at the component level if we are to know what types of errors to look for and what questions to ask.

This discussion will focus on the software components of a typical Web-based system:

- Client-based components on the front end, such as Web browsers, plug-ins, and embedded objects
- Server-side components on the back end, such as application server components, database applications, third-party modules, and cross-component communication

It offers insight to what typically happens when users click buttons on browser-based interfaces. It also explores pertinent testing questions such as:

- Which types of plug-ins are used by the application under test?
- What are the testing implications associated with these plug-ins?
- What issues should be considered during functionality and compatibility testing once the plug-ins have been integrated into the system?
- How should the distribution of server-side components affect test design and strategy?
- Which Web and database servers are supported by the application?
- How Web-to-database connectivity is implemented and what are the associated testing implications?
- How can testing be partitioned to focus on problematic components?

10.2.2 Distributed Application Architecture

In a distributed architecture, components are grouped into clusters of related services. Distributed architectures are used for both traditional client-server systems and Internet-based client-server systems.

10.2.2.1 Traditional Client-Server Systems

A database access application typically consists of four elements:

- User interface (UI) code – the end-user or input/output (I/O) devices interact with this for I/O operations.
- Business logic code – applies rules, computes data, and manipulates data.
- Data-access service code – handles data retrieval and updates to the database, in addition to sending results back to the client.
- Data storage – holds the information.

10.2.2.2 Thin- versus Thick-Client Systems

When the majority of processing is executed on the server-side, a system is considered to be a thin-client system. When the majority of processing is executed on the client-side, a system is considered to be a thick-client system.

In a thin-client system, the user interface runs on the client host while all other components run on the server host(s). The server is responsible for all services. After retrieving and processing data, only a plain HTML page is sent back to the client.

By contrast, in a thick-client system, most processing is done on the client-side; the client application handles data processing and applies logic rules to data. The server is responsible only for providing data access features and data storage. Components such as ActiveX controls and Java applets, which are required for the client to process data, are hosted and executed on the client machine.

Each of these systems calls for a different testing strategy. In thick-client systems, testing should focus on performance and compatibility. If Java applets are used, the applets will be sent to the browser with each request, unless the same applet is used within the same instance of the browser. If the applet is a few hundred kilobytes in size, it will take a fair amount of bandwidth to download it with reasonable response time.

Compatibility issues in thin-client systems are less of a concern. Performance issues do, however, need to be considered on the server-side, where requests are processed, and on the network where data transfer takes place (sending bitmaps to the browser). The thin-client system is designed to solve incompatibility problems as well as processing-power limitations on the client-side. Additionally, thin-client systems ensure that updates happen immediately, because the updates are applied at that server only.

Personal Digital Assistants (PDAs), for example, due to their small size, are not capable of handling much processing. The thin-client model serves PDAs well because it pushes the work to servers, which perform the processing and return results back to the client, or the PDA.

Desktop computers in which the operating systems deliver a lot of power and processing enable much more processing to be executed locally; therefore, the thick-client approach is commonly employed to improve overall performance.

10.2.3 Wireless Technologies

Wireless technologies represent a rapidly emerging area of growth and importance by providing ever-present access to the Internet and e-mail. It seems everyone wants unrestrained network access from general-purpose information to high-bandwidth applications. Even in the education world there is interest in creating mobile computing labs utilizing laptop computers equipped with wireless Ethernet cards.

The IT industry has made significant progress in resolving some issues to the widespread adoption of wireless technologies. Some of those issues have included non-existent standards, low bandwidth, and high infrastructure and service cost. Wireless is being adopted for many new applications: to connect computers, to allow remote monitoring and data acquisition, to provide access control and security, and to provide a solution for environments where wires may not be the best solution.

10.2.3.1 Important Issues for Wireless

As with any relatively new technology, there are many issues that affect implementation and utilization of wireless networks. There are both common and specific issues depending on the type of wireless network. Some of the common issues include: electromagnetic interference and physical obstacles that limit coverage of wireless networks. Some of the more specific issues are: standards, data security, throughput, and ease of use. The following issues are discussed further:

- Standards
- Coverage
- Security

10.2.3.1.1 Standards

A major obstacle for deployment of wireless networks is the existence of multiple standards. While GSM is the only widely supported standard in Europe and Asia, multiple standards are in use in the U.S. As a result, the U.S. has lagged in wireless networks deployment. Just recently, organizations have been formed to ensure network and device interoperability. For example, the adoption of the 802.11g standard has made wireless data networks one of the hottest newcomers in the current wireless market.

10.2.3.1.2 Coverage

Coverage mainly depends on the output power of the transmitter, its location and frequency used to transmit data. For example, lower frequencies are more forgiving when it comes to physical obstacles, like walls and stairways, while high frequencies require clear line of sight. For each particular application, throughput decreases as distance from the transmitter or access point increases.

10.2.3.1.3 Security

Data security is a major issue for wireless due to the nature of the transmission mechanism, which are the electromagnetic signals passing through the air. It is commonly believed that voice applications are less secure than data applications. This is due to limited capabilities of existing technologies to protect information that is being transmitted. For example, in metropolitan areas, users are at risk in that simple scanning devices can highjack cell phone numbers and be maliciously used.

In WLANs, authentication and encryption provide data security. Current implementations include:

- MAC address-based access lists on access points, where only registered and recognized MAC addresses are accepted and allowed to join the network.
- A closed wireless system, where users have to know the non-advertised network name to join.
- RADIUS server-based authentication, where users are authenticated against a centralized RADIUS server based on their MAC address or their user name and password.
- Wireless Equivalency Privacy (WEP) utilizes data encryption with 40-bit or 128-bit keys that are hidden from users. WEP provides three options, depending on the level of security needed: no encryption of data, combination of encrypted and non-encrypted data, and forced data encryption.
- High security solutions for encryption are proprietary. Both the Cisco AP-350 and Lucent/Agere AS-2000 offer per user/per session encryption keys and authenticates users based on a user name/password scheme.

It is important to understand that in WLANs, data is encrypted only between the wireless adapter and the access point. Data travels through a wired LAN un-encrypted. Therefore, data transmitted by wireless is not more secure than data transmitted through the wire, but probably not less secure. Application-level encryption mechanisms, like secure web transactions SSL and SSH, are responsible for further protection of data.

10.2.4 New Application Business Models

10.2.4.1 e-Commerce

The best way to look at e-commerce is to think of it as the conducting of business communication and transactions over networks and through computers. Some may even define electronic commerce (e-commerce) as the buying and selling of goods and services, and the transfer of funds, through digital communications. However e-commerce may also include all inter-company and intra-company functions (such as marketing, finance, manufacturing, selling, and negotiation) that enable commerce and use electronic mail, file transfer, fax, video conferencing, workflow, or interaction with a remote computer. Electronic commerce also includes buying and selling over the Web, electronic funds transfer, smart cards, digital cash, and all other ways of doing business over digital networks

10.2.4.2 e-Business

Electronic business (e-business) is, in its simplest form, the conduct of business on the Internet. It is a more generic term than e-commerce because it refers to not only buying and selling but also servicing customers and collaborating with business partners.

IBM, in 1997, was one of the first to use the term when it launched a campaign built around the term. Today, many corporations are rethinking their businesses in terms of the Internet and its capabilities. Companies are using the Web to buy parts and supplies from other companies, to collaborate on sales promotions, and to do joint research. Exploiting the convenience, availability, and global reach of the Internet, many companies, both large and small have already discovered how to use the Internet successfully.

Almost any organization can benefit from doing electronic business over the Internet. A larger company might launch a Web site as an extra marketing channel to increase sales by a few per cent. An individual with a computer and a modem or high speed cable might build an Internet-only business from nothing.

But you don't actually have to sell products from a Web site to benefit from Internet technology. For example, a factory could use the Web to make production information instantly available to sales offices hundreds of miles away, quite apart from selling products.

It doesn't take too much surfing on the Web to realize that most business sites today are little more than pretty brochures. They don't really do anything; they allow window-shopping only. But this, too, is e-business. Companies spend thousands of dollars perfecting their company image. By building better brand awareness, they hope to increase demand for their products.

10.2.5 New Communication Methods

10.2.5.1 Wireless Applications

There are numerous applications for all the different wireless technologies. To gain a better understanding the applications of wireless technologies are divided into the following groups:

- Voice and messaging
- Hand-held and other Internet-enabled devices
- Data networking

10.2.5.1.1 Voice and Messaging

Cell phones, pagers, and commercial two-way business radios can provide voice and messaging services. These devices will use either analog or digital standards that differ primarily in the way in which they process signals and encode information.

- The analog standard is the Advanced Mobile Phone Service (AMPS).
- The digital standards are: Global System for Mobile Communications (GSM), Time Division Multiple Access (TDMA), or Code Division Multiple Access (CDMA).

Normally, devices operate within networks that provide local, statewide, national or worldwide coverage. In the US, these large and costly networks are operated by carriers such as Sprint, Verizon, Cingular and local phone companies. They operate in different frequency bands allocated by the Federal Communications Commission (FCC).

Throughput depends on the standard being used, but presently in the U.S., these networks operate throughput rates up to 16 kilobits-per-second (Kbps). New digital standards, also referred to as Third-Generation Services or 3G, provide 30-times faster transfer rates and enhanced capabilities. Interoperability issues exist between networks, carriers, and devices because of the many standards. Generally, carriers charge their customers based on per minute utilization or per number of messages.

10.2.5.1.2 Hand-Held and Internet-Enabled Devices

Internet-enabled cell phones and Personal Digital Assistants (PDAs) have emerged as the newest products that can connect to the Internet across a digital wireless network. Available protocols, such as Wireless Application Protocol (WAP), and languages, such as WML (Wireless Markup Language) have been developed specifically for these devices to connect to the Internet.

However, the majority of current Internet content is not optimized for these devices; presently, e-mail, stock quotes, news, messages, and simple transaction-oriented services are optimally suited. Other limitations include:

- low bandwidth
- low quality of service
- high cost
- the need for additional equipment
- high utilization of devices' battery power

Nevertheless, this type of wireless technology is growing rapidly with better and more interoperable products.

10.2.5.1.3 Data Networking

This area focuses on pure data applications in wireless local area networks (WLANs) and data, voice, and video converged in broadband wireless. Also addressed briefly is Bluetooth, an emerging wireless technology.

- **Wireless Local Area Networks**

  Wireless Local Area Networks (WLAN) are implemented as an extension to wired LANs within a building and can provide the final few meters of connectivity between a wired network and the mobile user.

  WLANs are based on the IEEE 802.11 standard. There are three physical layers for WLANs: two radio frequency specifications (RF - direct sequence and frequency hopping spread spectrum) and one infrared (IR). Most WLANs operate in the 2.4 GHz license-free frequency band and have throughput rates up to 2 Mbps. The older 802.11b standard is direct sequence only, and provides throughput rates between 2 Mbps to 54 Mbps.

Currently the predominant standard is widely supported by vendors such as Cisco, Lucent, Apple, etc. The new standard, 802.11g, will operate in the 5 GHz license-free frequency band and is expected to provide throughput rates between 54 Mbps to 108 Mbps.

WLAN configurations vary from simple, independent, peer-to-peer connections between a set of PCs, to more complex, intra-building infrastructure networks. There are also point-to-point and point-to-multipoint wireless solutions. A point-to-point solution is used to bridge between two local area networks, and to provide an alternative to cable between two geographically distant locations (up to 30 miles). Point-to-multipoint solutions connect several, separate locations to one single location or building.

In a typical WLAN infrastructure configuration, there are two basic components:

- Access Points

  An access point/base station connects to a LAN by means of Ethernet cable. Usually installed in the ceiling, access points receive, buffer, and transmit data between the WLAN and the wired network infrastructure. A single access point supports on average twenty users and has a coverage varying from 60+ feet in areas with obstacles (walls, stairways, elevators) and up to 300+ feet in areas with clear line of sight. A building may require several access points to provide complete coverage and allow users to roam seamlessly between access points.

- Wireless Client Adapter

  A wireless adapter connects users via an access point to the rest of the LAN. A wireless adapter can be a PC card in a laptop, a USB adapter or a PCI adapter in a desktop computer, or can be fully integrated within a handheld device.

- **Broadband Wireless**

  Broadband wireless (BW) is an emerging wireless technology that allows simultaneous wireless delivery of voice, data, and video. BW is considered a competing technology with Digital Subscriber Line (DSL). It is generally implemented in metropolitan and urban areas and requires clear line of sight between the transmitter and the receiving end. BW provides the following two services, both of which operate in FCC-licensed frequency bands:

  - Local multi-point distribution service (LMDS)
  - Multi-channel multi-point distribution service (MMDS)

  LMDS is a high-bandwidth wireless networking service in the 28-31 GHz range of the frequency spectrum and has sufficient bandwidth to broadcast all the channels of direct broadcast satellite TV, all of the local over-the-air channels, and high speed full duplex data service. Average distance between LMDS transmitters is approximately one mile apart.

MMDS operates at lower frequencies, in the 2 GHz licensed frequency bands. MMDS has wider coverage than LMDS, up to 35 miles, but has lower throughput rates. Companies such as Sprint and Verizon own MMDS licenses in the majority of U.S. metropolitan areas. Broadband wireless still involves costly equipment and infrastructures. However, as it is more widely adopted, it is expected that the service cost will decrease.

- **Bluetooth**

  Bluetooth is a technology specification for small form factor, low-cost, short-range wireless links between mobile PCs, mobile phones, and other portable handheld devices, and connectivity to the Internet. The Bluetooth Special Interest Group (SIG) is driving development of the technology and bringing it to market and includes promoter companies such as, 3Com, Ericsson, IBM, Intel, Lucent, Motorola, Nokia, and over 1,800 Adopter/Associate member companies.

  Bluetooth covers a range of up to 30+ feet in the unlicensed 2.4GHz band. Because 802.11 WLANs also operate in the same band, there are interference issues to consider. Although Bluetooth technology and products started being available in 2001, interoperability still seems to be a big problem.

10.2.6 New Testing Tools

10.2.6.1 Test Automation

Test automation is the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.

Over the past few years, tools that help developers quickly create applications with graphical user interfaces (GUI) have dramatically improved their productivity. This improved productivity has increased the pressure on testers, who are often perceived as bottlenecks to the delivery of software products. Testers are being asked to test more and more code in less and less time. Test automation is one way to help testers quickly create and execute tests, as manual testing is time consuming.

New test automation tools are now available that help testers dramatically improve their productivity by reducing the test time as well as the cost. Other test automation tools support execution of performance tests. Many test automation tools provide record and playback features that allow users to record interactively user actions and replay it back any number of times, comparing actual results to those expected.

A growing trend in software development is to use testing frameworks, such as the JUnit or NUnit frameworks. These allow the code to conduct unit tests to determine whether various sections of the code are acting as expected in various circumstances. Test cases describe tests

that need to be run on the program to verify that the program runs as expected. In a properly tested program, there is at least test in the code designed to satisfy a unit test requirement.

JUnit is a unit test framework for the Java programming language. JUnit was created by Kent Beck along with Erich Gamma. Since then, JUnit has served as an inspiration and a role model for a variety of other unit test frameworks for other languages, such as (in alphabetical order):

- AsUnit for ActionScript
- NUnit for Microsoft.Net
- PHPUnit for PHP
- PyUnit for Python
- SimpleTest for PHP
- Test:Class for Perl (among others)

Experience gained with JUnit has been important in the development of test-driven development. As a result, some knowledge of JUnit is often presumed in discussions of test-driven development as can be seen in the book, Test-Driven Development: By Example, by Kent Beck.

Another important aspect of test automation is the idea of partial test automation, or automating parts but not all of the software testing process. If, for example, an oracle cannot reasonably be created, or if fully automated tests would be too difficult to maintain, then a software tools engineer can create testing tools to help testers perform their jobs more efficiently.

Testing tools can help automate tasks without necessarily automating tests in an end-to-end fashion such as in the following tasks:

- product installation
- test data creation
- GUI interaction
- problem detection (consider parsing or polling agents equipped with oracles)
- defect logging

10.3 Testing the Effectiveness of Integrating New Technology

The mission assigned to software testers will determine whether or not testers need to assess the impact of new technologies on their software testing roles and responsibilities. That responsibility can be assigned to software testers, software developers or process engineering and quality assurance groups. If the responsibility is assigned to testers they need to develop the competencies to fulfill that responsibility.

This skill category has been included in the software testing CBOK because it is a testing activity performed by many software testing groups. It also represents an opportunity for software testers to provide more value to their IT organizations. The material in this skill category has been designed to define a potential role for software testers and provide insight into how that responsibility might be fulfilled.

This integration effectiveness test involves these two tasks:

- Determine the process maturity level of the technology
- Test the adequacy of the controls over implementing the new technology

10.3.1 Determine the Process Maturity Level of the Technology

The high productivity increases associated with the use of IT are partially attributable to the new technologies organizations use. Since the early days of computing the computing power of systems increases ten-fold every few years. New supporting work processes and methods are needed to benefit from that change in technology.

Any change has risk associated with it. However, those risks should not discourage organizations from acquiring and implementing new technologies. What is important is that the appropriate infrastructure and processes are in place to control the use of the new technologies effectively and efficiently.

The SEI capability maturity model uses five levels of maturity. The model shows a normal evolution from a state that many refer to as “chaos” which is Level 1 to a very mature level in which the use of technology and processes is optimized.

Each time new technology is acquired an IT organization’s ability to optimize the use of that technology reverts to Level 1. The maturity process needs to be repeated to mature the use of that technology until it is optimized into the organization’s systems. Depending on the technology acquired the time span to mature the use of that technology can vary from days to years.

The maturing of technology occurs through building and implementing work processes. At lower levels there is significant variance in performing work using new technologies while at higher levels that variance is significantly reduced. For example if an IT organization acquired an agile process for building software one would expect that shortly after acquiring the agile technology the implementation effort to implement like projects would vary significantly from project to project. As the agile technology is implemented into the work processes and training programs, that variance is reduced so that at higher maturity levels the effort to implement like projects should be very similar.

The following is a version of SEI’s capability maturing model that focuses on maturing the use of the newer IT technologies. Testers should have a high-level knowledge of how to mature a new technology so they can determine the level of risk associated with a specific technology. Knowing the maturity level of a new technology should aid in building a test plan that focuses on the level of risk, testers must address when building their test plan.

10.3.1.1 Level 1 – People-Dependent Technology

At Level 1 the technology is available for use by individuals at their discretion. Generally they are not required to use the technology, nor are there specific defined methods for using the technology. The individual must understand the technology, become competent in its use, and utilize it for building and operating information systems. The effectiveness and efficiency of technology at Level 1 is solely people-dependent.

10.3.1.2 Level 2 – Use Description-Dependent Technology Processes

At Level 2 there are processes in how to use the technology, but projects using that technology can, at their discretion, vary from the process. For example, security "handshaking" routines are proposed for wireless technology, but if project leaders do not believe that level of security is needed, they can implement the system without the security. Level 2 recognizes the processes are defective and enables project personnel to make changes they deem appropriate.

10.3.1.3 Level 3 – Use of Technology

At this level the processes are determined to be effective and compliance to those processes are required. Any variance from the process must be approved by management. Normally quality control procedures are in place to check compliance and report variance from the process. For example, if security "handshaking" procedures were required for wireless technology no wireless system could be implemented without those handshaking processes in place.

10.3.1.4 Level 4 – Quantitatively Measured Technology

Once the processes for using new technology are deemed effective, they should be measured. Generally measurement for process optimization is not effective until the processes are stabilized and complied with. When they are stabilized and complied with the quantitative data collected is reliable. Measurement theory states that until measurement data is reliable it cannot be used to optimize the work processes.

10.3.1.5 Level 5 – Optimized Use of Technology

At this level the quantitative data produced at Level 4 will enable the optimization of the use of technology. Variance is reduced to close to zero, defects are minimal, and the organization has maximized the use of technology. Generally by the time technology has matured to Level 5, new technology is available which moves the technology maturity level back to Level 1.

10.3.2 Test the Controls over Implementing the New Technology

Testing the adequacy of the controls over the new technology to evaluate the effectiveness of the implementation involves these three tasks:

- Testing actual performance versus stated performance

  Does the technology accomplish its stated benefits?

- Test the adequacy of the current processes to control the technology

  Do implemented work processes assure the technology will be used in an effective and efficient manner?

- Assess whether staff skills are adequate to effectively use the technology

  Are the users of the technology sufficiently trained to use the technology effectively?

10.3.2.1 Test Actual Performance versus Stated Performance

Software systems are built based on assumptions. For example, the assumptions that:

- hardware technology can perform at predefined speeds
- the staff assigned to a project can do the defined tasks
- if a specific work process is followed it will produce a predefined desired result

With currently implemented technology there is experience to support these assumptions. For example, if a specific individual is to use a specific developmental or testing tool, and those who have graduated from a specified training class can perform tasks using those tools, it is reasonable to assume that if project personnel had that training they can perform the defined work tasks.

When new technologies are introduced there are no experiences to support project assumptions. For example, if a software package was stated to search a database and retrieve desired information within X% of a second, that assumption would have to be tested. If the software failed to meet the performance specifications, a software system objective might not be met. For example if a system objective was that a customer online to the organization's database could determine the availability of a specific product within two seconds, and the hardware and software used to make that determination took 15 seconds the customer might leave the Web site and the sale would be lost.

The following is a list of some of the tests that testers may want to perform to evaluate whether or not a new technology can or cannot meet the specified performance criteria:

- Documentation represents actual technology execution.
- Training courses transfer the needed knowledge to use the technology.
- New technology is compatible with existing technology.
- Stated performance criteria represent actual performance criteria.
- Promised vendor support equals actual vendor support.

- Expected test processes and tools are effective in testing new technologies.

10.3.2.2 Test the Adequacy of the Current Processes to Control the Technology

The underlying premise of process engineering is that the method the IT organization performs to do work will be controlled. Work processes are the means for controlling the use of new technologies. Work processes include these three process attributes:

- Standard

  This is the objective to be achieved using new technology.

- Procedure

  The step-by-step methods that will be performed to utilize the new technology.

- Quality Control

  The means that will be used to determine that the procedures to do work are performed in the manner specified in those procedures.

If new technologies are to be used effectively and efficiently the work processes governing the use of those technologies must include the standard, the procedures and the quality control procedures. Without an objective the workers do not know the purpose for using the technology, without procedures they do not know how to use the technology, without quality control they do not know if they used the procedures correctly.

If the new technology lacks any of these three important process engineering components additional testing should be performed to assess the risk associated with using that technology:

- Process

  Requires the three process attributes: Standard, Procedure and Quality Control

- Compliance

  Management's desire that workers follow the procedures to do work and procedures to check work.

- Enforcement

  If workers do not follow the appropriate work procedures management needs to take action to enforce compliance to those procedures.

Any new technology that is deemed "out of control" requires additional attention. Out of control meaning any or all of the key process engineering components are not present in the use of that technology. Out of control from a test perspective means that testers cannot make assumptions that work will be performed as specified, and the software will not execute as specified.

If the testers determine that processes are not adequate to control new technology, they can take any one of the following three actions:

- Identify the new technology risk and report that risk to the appropriate software system stakeholder.
- Identify the potentially ineffective parts of the work process as related to that new technology. This might include presenting the technology maturity level and/or any of the missing key process engineering components.
- Conduct tests to identify specific problems uncovered associated with the new technology, and assessing the potential impact of that problem on the operational system.

10.3.3 Test the Adequacy of Staff Skills to Use the Technology

There are two components to any professional process. The components are the process itself, and the competency of the process user to use that process effectively. The two components must be properly coordinated for the new technology governed by that process to be used effectively.

Figure 10-1 shows the relationship of the competency of the user technology to the process controlling the use of that technology. The figure shows the five levels of technology process maturity, and the competency of the user to use the technology from high to low.

The key concept of Figure 10-1 is when using technology do you rely more on the competency of the user or the effectiveness of the process. When the technology process maturity is low there is greater reliance on the competency of the user. When the technology process maturity is high there is more reliance on the process effectiveness and less reliance on the user competency.

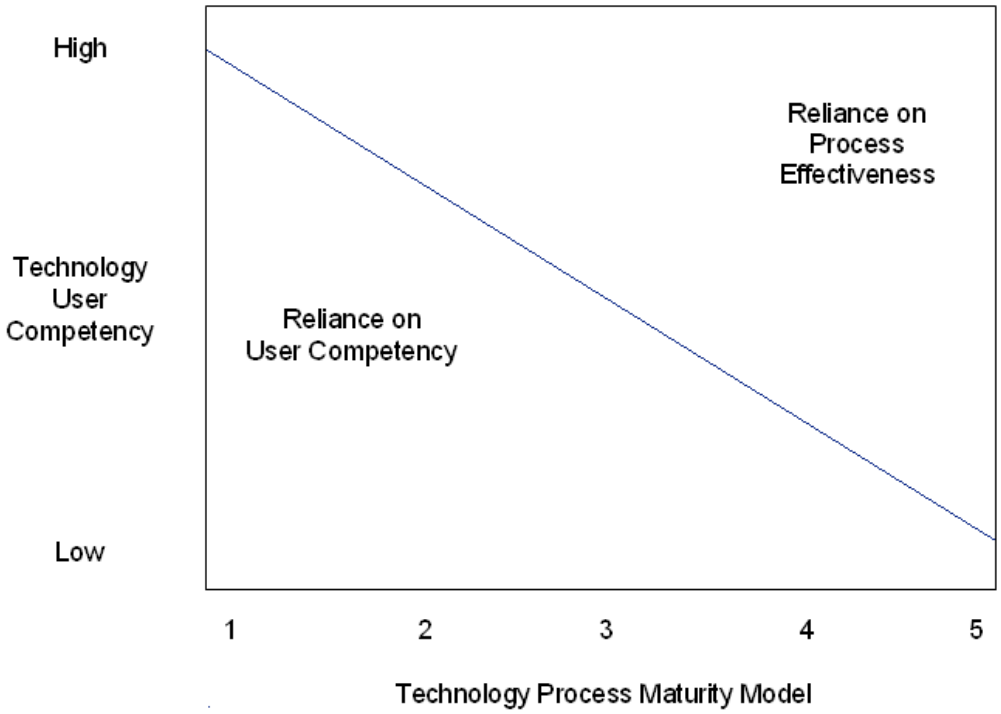


Figure 10-1 Relationship of Competency to Process

The reason for the changing reliance between user and process is transfer of knowledge. At technical process maturity Level 1 most of the ability to use new technology effectively resides with the user. As the user(s) learns to use the technology more effectively there is a knowledge transfer from the user to the process. For example, if the user of an agile software development process learned that agile technology was only effective with small software systems, that knowledge could be incorporated into the agile software development process. For example, prior to using the process the user would assess the size of the system and, if it was large, use another software development methodology to build the software.

In assessing the adequacy of the staff competency to use new technology the tester should evaluate:

- Technological process maturity level

  At lower levels more user competency is needed than at higher levels.

- Training available in the new technology

  Testers need to determine the type of training available, the experience of the effectiveness of that training and whether or not the users of the new technology have gone through that training.

- The performance evaluation of individuals to use the new technology effectively

  This looks at the individual's past experience while working with that technology on projects and then assessing the effectiveness of the projects in using that technology.

Appendix A

Vocabulary

Access Modeling Used to verify that data requirements (represented in the form of an entity-relationship diagram) support the data demands of process requirements (represented in data flow diagrams and process specifications.)

Affinity Diagram A group process that takes large amounts of language data, such as a list developed by brainstorming, and divides it into categories.

Application A single software product that may or may not fully support a business function.

Audit This is an inspection/assessment activity that verifies compliance with plans, policies, and procedures, and ensures that resources are conserved. Audit is a staff function; it serves as the "eyes and ears" of management.

Backlog Work waiting to be done; for IT this includes new systems to be developed and enhancements to existing systems. To be included in the development backlog, the work must have been cost-justified and approved for development.

Baseline A quantitative measure of the current level of performance.

Benchmarking Comparing your company's products, services, or processes against best practices, or competitive practices, to help define superior performance of a product, service, or support process.

Benefits Realization Test A test or analysis conducted after an application is moved into production to determine whether it is likely to meet the originating business case.

Black-Box Testing A test technique that focuses on testing the functionality of the program, component, or application against its specifications

without knowledge of how the system is constructed; usually data or business process driven.

Boundary Value Analysis A data selection technique in which test data is chosen from the "boundaries" of the input or output domain classes, data structures, and procedure parameters. Choices often include the actual minimum and maximum boundary values, the maximum value plus or minus one, and the minimum value plus or minus one.

Brainstorming A group process for generating creative and diverse ideas.

Branch Testing
A test method that requires that each possible branch on each decision point be executed at least once.

Bug A general term for all software defects or errors.

Candidate An individual who has met eligibility requirements for a credential awarded through a certification program, but who has not yet earned that certification through participation in the required skill and knowledge assessment instruments.

Cause-Effect Graphing A tool used to derive test cases from specifications. A graph that relates causes (or input conditions) to effects is generated. The information in the graph is converted into a decision table where the columns are the cause-effect combinations. Unique rows represent test cases.

Certificant An individual who has earned a credential awarded through a certification program.

Certification A voluntary process instituted by a nongovernmental agency by which individual applicants are recognized for having achieved a measurable level of skill or knowledge. Measurement of the skill or knowledge makes certification more restrictive than simple registration, but much less restrictive than formal licensure.

Checklists A series of probing questions about the completeness and attributes of an application system. Well-constructed checklists cause evaluation of areas, which are prone to problems. It both limits the scope of the test and directs the tester to the areas in which there is a high probability of a problem.

Checkpoint Review Held at predefined points in the development process to evaluate whether certain quality factors (critical success factors) are being adequately addressed in the system being built. Independent experts for the purpose of identifying problems conduct the reviews as early as possible.

Checksheet A form used to record data as it is gathered.

Client The customer that pays for the product received and receives the benefit from the use of the product.

Coaching Providing advice and encouragement to an individual or individuals to promote a desired behavior.

Code Comparison One version of source or object code is compared to a second version. The objective is to identify those portions of computer programs that have been changed. The technique is used to identify those segments of an application program that have been altered as a result of a program change.

Compiler-Based Analysis Most compilers for programming languages include diagnostics that identify potential program structure flaws. Many of these diagnostics are warning messages requiring the programmer to conduct additional investigation to determine whether or not the problem is real. Problems may include syntax problems, command violations, or variable/data reference problems. These diagnostic messages are a useful means of detecting program problems, and should be used by the programmer.

Complete Test Set A test set containing data that causes each element of pre-specified set of Boolean conditions to be true. In addition, each element of the test set causes at least one condition to be true.

Completeness The property that all necessary parts of an entity are included. Often, a product is said to be complete if it has met all requirements.

Complexity-Based Analysis Based upon applying mathematical graph theory to programs and preliminary design language specification (PDLs) to determine a unit's complexity. This analysis can be used to measure and control complexity when maintainability is a desired attribute. It can also be used to estimate test effort required and identify paths that must be tested.

Compliance Checkers A parse program looking for violations of company standards. Statements that contain violations are flagged. Company standards are rules that can be added, changed, and deleted as needed.

Condition Coverage A white-box testing technique that measures the number of, or percentage of, decision outcomes covered by the test cases designed. 100% condition coverage would indicate that every possible outcome of each decision had been executed at least once during testing.

Configuration Management Tools Tools that are used to keep track of changes made to systems and all related artifacts. These are also known as version control tools.

Configuration Testing Testing of an application on all supported hardware and software platforms. This may include various combinations of hardware types, configuration settings, and software versions.

Consistency The property of logical coherence among constituent parts. Consistency can also be expressed as adherence to a given set of rules.

Consistent Condition Set A set of Boolean conditions such that complete test sets for the conditions uncover the same errors.

Control Flow Analysis

Based upon graphical representation of the program process. In control flow analysis, the program graph has nodes, which represent a statement or segment possibly ending in an unresolved branch. The graph illustrates the flow of program control from one segment to another as illustrated through branches. The objective of control flow analysis is to determine potential problems in logic branches that might result in a loop condition or improper processing.

Conversion Testing Validates the effectiveness of data conversion processes, including field-to-field mapping, and data translation.

Correctness The extent to which software is free from design and coding defects (i.e., fault-free). It is also the extent to which software meets its specified requirements and user objectives.

Cost of Quality (COQ) Money spent beyond expected production costs (labor, materials, equipment) to ensure that the product the customer receives is a quality (defect free) product. The Cost of Quality includes prevention, appraisal, and correction or repair costs.

Coverage-Based Analysis A metric used to show the logic covered during a test session, providing insight to the extent of testing. The simplest metric for coverage would be the number of computer statements executed during the test compared to the total number of statements in the program. To completely test the program structure, the test data chosen should cause the execution of all paths. Since this is not generally possible outside of unit test, general metrics have been developed which give a measure of the quality of test data based on the proximity to this ideal coverage. The metrics should take into consideration the existence of infeasible paths, which are those paths in the program that have been designed so that no data will cause the execution of those paths.

Customer The individual or organization, internal or external to the producing organization that receives the product.

Cyclomatic Complexity

The number of decision statements, plus one.

Data Dictionary Provides the capability to create test data to test validation for the defined data elements. The test data generated is based upon the attributes defined for each data element. The test data will check both the normal variables for each data element as well as abnormal or error conditions for each data element.

DD (Decision-to-Decision) Path A path of logical code sequence that begins at a decision statement or an entry and ends at a decision statement or an exit.

Debugging The process of analyzing and correcting syntactic, logic, and other errors identified during testing.

Decision Coverage A white-box testing technique that measures the number of, or percentage of, decision directions executed by the test case designed. 100% decision coverage would indicate that all decision directions had been executed at least once during testing. Alternatively, each logical path through the program can be tested. Often, paths through the program are grouped into a finite set of classes, and one path from each class is tested.

Decision Table A tool for documenting the unique combinations of conditions and associated results in order to derive unique test cases for validation testing.

Defect Operationally, it is useful to work with two definitions of a defect:

- From the producer's viewpoint a defect is a product requirement that has not been met or a product attribute possessed by a product or a function performed by a product that is not in the statement of requirements that define the product;
- From the customer's viewpoint a defect is anything that causes customer dissatisfaction, whether in the statement of requirements or not.

Defect Tracking Tools Tools for documenting defects as they are found during testing and for tracking their status through to resolution.

Design Level The design decomposition of the software item (e.g., system, subsystem, program, or module).

Desk Checking The most traditional means for analyzing a system or a program. Desk checking is conducted by the developer of a system or program. The process involves reviewing the complete product to ensure that it is structurally sound and that the standards and requirements have been met. This tool can also be used on artifacts created during analysis and design.

Driver Code that sets up an environment and calls a module for test.

Dynamic Analysis Analysis performed by executing the program code. Dynamic analysis executes or simulates a development phase product, and it detects errors by analyzing the response of a product to sets of input data.

Dynamic Assertion A dynamic analysis technique that inserts into the program code assertions about the relationship between program variables. The truth of the assertions is determined as the program executes.

Empowerment Giving people the knowledge, skills, and authority to act within their area of expertise to do the work and improve the process.

Entrance Criteria Required conditions and standards for work product quality that must be present or met for entry into the next stage of the software development process.

Equivalence Partitioning The input domain of a system is partitioned into classes of representative values so that the number of test cases can be limited to one-per-class, which represents the minimum number of test cases that must be executed.

Error or Defect A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

Human action that results in software containing a fault (e.g., omission or misinterpretation of user requirements in a software specification, incorrect translation, or omission of a requirement in the design specification).

Error Guessing Test data selection technique for picking values that seem likely to cause defects. This technique is based upon the theory that test cases and test data can be developed based on the intuition and experience of the tester.

Exhaustive Testing Executing the program through all possible combinations of values for program variables.

Exit Criteria Standards for work product quality, which block the promotion of incomplete or defective work products to subsequent stages of the software development process.

File Comparison Useful in identifying regression errors. A snapshot of the correct expected results must be saved so it can be used for later comparison.

Flowchart Pictorial representations of data flow and computer logic. It is frequently easier to understand and assess the structure and logic of an application system by developing a flow chart than to attempt to understand narrative descriptions or verbal

explanations. The flowcharts for systems are normally developed manually, while flowcharts of programs can be produced.

Force Field Analysis A group technique used to identify both driving and restraining forces that influence a current situation.

Formal Analysis Technique that uses rigorous mathematical techniques to analyze the algorithms of a solution for numerical properties, efficiency, and correctness.

Functional Testing Application of test data derived from the specified functional requirements without regard to the final program structure.

Histogram A graphical description of individually measured values in a data set that is organized according to the frequency or relative frequency of occurrence. A histogram illustrates the shape of the distribution of individual values in a data set along with information regarding the average and variation.

Infeasible Path A sequence of program statements that can never be executed.

Inputs Materials, services, or information needed from suppliers to make a process work, or build a product.

Inspection A formal assessment of a work product conducted by one or more qualified independent reviewers to detect defects, violations of development standards, and other problems. Inspections involve authors only when specific questions concerning deliverables exist. An inspection identifies defects, but does not attempt to correct them. Authors take corrective actions and arrange follow-up reviews as needed.

Instrumentation The insertion of additional code into a program to collect information about program behavior during program execution.

Integration Testing This test begins after two or more programs or application components have been successfully unit tested. It is conducted by the development team to validate the technical quality or design of the application. It is the first level of testing which formally integrates a set of programs that communicate among themselves via messages or files (a client and its server(s), a string of batch programs, or a set of online modules within a dialog or conversation.)

Invalid Input Test data that lays outside the domain of the function the program represents.

Leadership The ability to lead, including inspiring others in a shared vision of what can be, taking risks, serving as a role model,

reinforcing and rewarding the accomplishments of others, and helping others to act.

Life Cycle Testing The process of verifying the consistency, completeness, and correctness of software at each stage of the development life cycle.

Management A team or individuals who manage(s) resources at any level of the organization.

Mapping Provides a picture of the use of instructions during the execution of a program. Specifically, it provides a frequency listing of source code statements showing both the number of times an instruction was executed and which instructions were not executed. Mapping can be used to optimize source code by identifying the frequently used instructions. It can also be used to determine unused code, which can demonstrate code, which has not been tested, code that is infrequently used, or code that is non-entrant.

Mean A value derived by adding several quantities and dividing the sum by the number of these quantities.
Metric-Based Test Data Generation
The process of generating test sets for structural testing based on use of complexity or coverage metrics.

Model Animation Model animation verifies that early models can handle the various types of events found in production data. This is verified by “running” actual production transactions through the models as if they were operational systems.

Model Balancing Model balancing relies on the complementary relationships between the various models used in structured analysis (event, process, data) to ensure that modeling rules/standards have been followed; this ensures that these complementary views are consistent and complete.

Mission A customer-oriented statement of purpose for a unit or a team.

Mutation Analysis A method to determine test set thoroughness by measuring the extent to which a test set can discriminate the program from slight variants (i.e., mutants) of it.

Network Analyzers A tool used to assist in detecting and diagnosing network problems.

Oracle A (typically automated) mechanism or principle by which a problem in the software can be recognized. For example, automated test oracles have value in load testing software (by signing on to an application with hundreds or thousands of

instances simultaneously), or in checking for intermittent errors in software.

Outputs Products, services, or information supplied to meet customer needs.

Pass/Fail Criteria Decision rules used to determine whether a software item or feature passes or fails a test.

Path Expressions A sequence of edges from the program graph that represents a path through the program.

Path Testing A test method satisfying the coverage criteria that each logical path through the program be tested. Often, paths through the program are grouped into a finite set of classes and one path from each class is tested.

Performance Test Validates that both the online response time and batch run times meet the defined performance requirements.

Performance/Timing Analyzer A tool to measure system performance.

Phase (or Stage) Containment A method of control put in place within each stage of the development process to promote error identification and resolution so that defects are not propagated downstream to subsequent stages of the development process. The verification, validation, and testing of work within the stage that it is created.

Policy Managerial desires and intents concerning either process (intended objectives) or products (desired attributes).

Population Analysis Analyzes production data to identify, independent from the specifications, the types and frequency of data that the system will have to process/produce. This verifies that the specs can handle types and frequency of actual data and can be used to create validation tests.

Procedure The step-by-step method followed to ensure that standards are met.

Process The work effort that produces a product. This includes efforts of people and equipment guided by policies, standards, and procedures.

The process or set of processes used by an organization or project to plan, manage, execute, monitor, control, and improve its software related activities. A set of activities and tasks. A statement of purpose and an essential set of practices (activities) that address that purpose.

Process Improvement To change a process to make the process produce a given product faster, more economically, or of higher quality. Such changes may require the product to be changed. The defect rate must be maintained or reduced.

Product The output of a process: the work product. There are three useful classes of products: Manufactured Products (standard and custom), Administrative/Information Products (invoices, letters, etc.), and Service Products (physical, intellectual, physiological, and psychological). A statement of requirements defines products; one or more people working in a process produce them.

Product Improvement To change the statement of requirements that defines a product to make the product more satisfying and attractive to the customer (more competitive). Such changes may add to or delete from the list of attributes and/or the list of functions defining a product. Such changes frequently require the process to be changed. Note: This process could result in a very new product.

Production Costs The cost of producing a product. Production costs, as currently reported, consist of (at least) two parts; actual production or right-the-first time costs (RFT) plus the Cost of Quality (COQ). RFT costs include labor, materials, and equipment needed to provide the product correctly the first time.

Productivity The ratio of the output of a process to the input, usually measured in the same units. It is frequently useful to compare the value added to a product by a process, to the value of the input resources required (using fair market values for both input and output).

Proof of Correctness The use of mathematical logic techniques to show that a relationship between program variables assumed true at program entry implies that another relationship between program variables holds at program exit.

Quality A product is a quality product if it is defect free. To the producer, a product is a quality product if it meets or conforms to the statement of requirements that defines the product. This statement is usually shortened to: quality means meets requirements. From a customer's perspective, quality means "fit for use."

Quality Assurance (QA) The set of support activities (including facilitation, training, measurement, and analysis) needed to provide adequate confidence that processes are established and continuously improved to produce products that meet specifications and are fit for use.

Quality Control (QC) The process by which product quality is compared with applicable standards, and the action taken when nonconformance is detected. Its focus is defect detection and removal. This is a line function; that is, the performance of these tasks is the responsibility of the people working within the process.

Quality Function Deployment (QFD) A systematic matrix method used to translate customer wants or needs into product or service characteristics that will have a significant positive impact on meeting customer demands.

Quality Improvement To change a production process so that the rate at which defective products (defects) are produced is reduced. Some process changes may require the product to be changed.

Recovery Test Evaluates the contingency features built into the application for handling interruptions and for returning to specific points in the application processing cycle, including checkpoints, backups, restores, and restarts. This test also assures that disaster recovery is possible.

Regression Testing Testing of a previously verified program or application following program modification for extension or correction to ensure no new defects have been introduced.

Requirement A formal statement of:

1. An attribute to be possessed by the product or a function to be performed by the product
2. The performance standard for the attribute or function;

   and/or
3. The measuring process to be used in verifying that the standard has been met.

Risk Matrix Shows the controls within application systems used to reduce the identified risk, and in what segment of the application those risks exist. One dimension of the matrix is the risk, the second dimension is the segment of the application system, and within the matrix at the intersections are the controls. For example, if a risk is “incorrect input” and the systems segment is “data entry,” then the intersection within the matrix would show the controls designed to reduce the risk of incorrect input during the data entry segment of the application system.

Run Chart A graph of data points in chronological order used to illustrate trends or cycles of the characteristic being measured to suggest an assignable cause rather than random variation.

Scatter Plot Diagram A graph designed to show whether there is a relationship between two changing variables.

Self-validating Code Code that makes an explicit attempt to determine its own correctness and to proceed accordingly.

Services See Product.

Simulation Use of an executable model to represent the behavior of an object. During testing, the computational hardware, the external environment, and even code segments may be simulated.

Software Feature A distinguishing characteristic of a software item (e.g., performance, portability, or functionality).

Software Item Source code, object code, job control code, control data, or a collection of these.

Special Test Data Test data based on input values that are likely to require special handling by the program.

Standardize Procedures that are implemented to ensure that the output of a process is maintained at a desired level.

Standards The measure used to evaluate products and identify nonconformance. The basis upon which adherence to policies is measured.

Statement of Requirements The exhaustive list of requirements that define a product. Note that the statement of requirements should document requirements proposed and rejected (including the reason for the rejection) during the requirement determination process.

Statement Testing A test method that executes each statement in a program at least once during program testing.

Static Analysis Analysis of a program that is performed without executing the program. It may be applied to the requirements, design, or code.

Statistical Process Control The use of statistical techniques and tools to measure an ongoing process for change or stability.

Stress Testing This test subjects a system, or components of a system, to varying environmental conditions that defy normal expectations. For example, high transaction volume, large database size or restart/recovery circumstances. The intention of stress testing is to identify constraints and to ensure that there are no performance problems.

Structural Testing A testing method in which the test data is derived solely from the program structure.

Stub Special code segments that when invoked by a code segment under testing, simulate the behavior of designed and specified modules not yet constructed.

Supplier An individual or organization that supplies inputs needed to generate a product, service, or information to a customer.

Symbolic Execution

A method of symbolically defining data that forces program paths to be executed. Instead of executing the program with actual data values, the variable names that hold the input values are used. Thus, all variable manipulations and decisions are made symbolically. This process is used to verify the completeness of the structure, as opposed to assessing the functional requirements of the program.

System One or more software applications that together support a business function.

System Test During this event, the entire system is tested to verify that all functional, information, structural and quality requirements have been met. A predetermined combination of tests is designed that, when executed successfully, satisfy management that the system meets specifications. System testing verifies the functional quality of the system in addition to all external interfaces, manual procedures, restart and recovery, and human-computer interfaces. It also verifies that interfaces between the application and the open environment work correctly, that JCL functions correctly, and that the application functions appropriately with the Database Management System, Operations environment, and any communications systems.

Test

1. A set of one or more test cases.
2. A set of one or more test cases and procedures.

Test Case Generator A software tool that creates test cases from requirements specifications. Cases generated this way ensure that 100% of the functionality specified is tested.

Test Case Specification An individual test condition, executed as part of a larger test that contributes to the test's objectives. Test cases document the input, expected results, and execution conditions of a given test item. Test cases are broken down into one or more detailed test scripts and test data conditions for execution.

Test Cycle Test cases are grouped into manageable (and schedulable) units called test cycles. Grouping is according to the relation of objectives to one another, timing requirements, and on the best way to expedite defect detection during the testing event. Often test cycles are linked with execution of a batch process.

Test Data Generator A software package that creates test transactions for testing application systems and programs. The type of transactions that can be generated is dependent upon the options available in the test data generator. With many current generators, the prime advantage is the ability to create a large number of transactions to volume test application systems.

Test Data Set Set of input elements used in the testing process.

Test Design Specification A document that specifies the details of the test approach for a software feature or a combination of features and identifies the associated tests.

Test Driver A program that directs the execution of another program against a collection of test data sets. Usually, the test driver also records and organizes the output generated as the tests are run.

Test Harness A collection of test drivers and test stubs.

Test Incident Report A document describing any event during the testing process that requires investigation.

Test Item A software item that is an object of testing.

Test Item Transmittal Report A document that identifies test items and includes status and location information.

Test Log A chronological record of relevant details about the execution of tests.

Test Plan A document describing the intended scope, approach, resources, and schedule of testing activities. It identifies test items, the features to be tested, the testing tasks, the personnel performing each task, and any risks requiring contingency planning.

Test Procedure Specification A document specifying a sequence of actions for the execution of a test.

Test Scripts A tool that specifies an order of actions that should be performed during a test session. The script also contains expected results. Test scripts may be manually prepared using paper forms, or may be automated using capture/playback tools or other kinds of automated scripting tools.

Test Stubs Simulates a called routine so that the calling routine's functions can be tested. A test harness (or driver) simulates a calling component or external environment, providing input to the called routine, initiating the routine, and evaluating or displaying output returned.

Test Suite Manager A tool that allows testers to organize test scripts by function or other grouping.

Test Summary Report A document that describes testing activities and results and evaluates the corresponding test items.

Tracing

A process that follows the flow of computer logic at execution time. Tracing demonstrates the sequence of instructions or a path followed in accomplishing a given task. The two main types of trace are tracing instructions in computer programs as they are executed, or tracing the path through a database to locate predetermined pieces of information.

Unit Test Testing individual programs, modules, or components to demonstrate that the work package executes per specification, and validate the design and technical quality of the application. The focus is on ensuring that the detailed logic within the component is accurate and reliable according to pre-determined specifications. Testing stubs or drivers may be used to simulate behavior of interfacing modules.

Usability Test The purpose of this event is to review the application user interface and other human factors of the application with the people who will be using the application. This is to ensure that the design (layout and sequence, etc.) enables the business functions to be executed as easily and intuitively as possible. This review includes assuring that the user interface adheres to documented User Interface standards, and should be conducted early in the design stage of development. Ideally, an application prototype is used to walk the client group through various business scenarios, although paper copies of screens, windows, menus, and reports can be used.

User The customer that actually uses the product received.

User Acceptance Test

User Acceptance Testing (UAT) is conducted to ensure that the system meets the needs of the organization and the end user/ customer. It validates that the system will work as intended by the user in the real world, and is based on real world business scenarios, not system requirements. Essentially, this test validates that the right system was built.

Valid Input Test data that lie within the domain of the function represented by the program.

Validation Determination of the correctness of the final program or software produced from a development project with respect to the user needs and requirements. Validation is usually accomplished by verifying each stage of the software development life cycle.

Values (Sociology) The ideals, customs, instructions, etc., of a society toward which the people have an affective regard. These values may be positive, as cleanliness, freedom, or education, or negative, as cruelty, crime, or blasphemy. Any object or quality desired as a means or as an end in itself.

Verification The process of determining whether the products of a given phase of the software development cycle fulfill the requirements established during the previous phase.
The act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether items, processes, services, or documents conform to specified requirements.

Vision A vision is a statement that describes the desired future state of a unit.

Walkthroughs During a walkthrough, the producer of a product "walks through" or paraphrases the products content, while a team of other individuals follow along. The team's job is to ask questions and raise issues about the product that may lead to defect identification.

White-Box Testing A testing technique that assumes that the path of the logic in a program unit or component is known. White-box testing usually consists of testing paths, branch by branch, to produce predictable results. This technique is usually used during tests executed by the development team, such as Unit or Component testing,

Appendix B

References

It is each candidate's responsibility to stay current in the field and to be aware of published works and materials available for professional study and development. Software Certifications recommends that candidates for certification continually research and stay aware of current literature and trends in the field. There are many valuable references that have not been listed here. These references are for informational purposes only.

Beck, Kent. *Test Driven Development: By Example.* Addison-Wesley Professional, First Edition, 2002.

Beizer, Boris. *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc., 1995.

Black, Rex. *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*. John Wiley & Sons, Inc., Second Edition, 2002.

Copeland, Lee. *A Practitioner's Guide to Software Test Design*. Artech House Publishers, 2003.

Craig, Rick D. *Systematic Software Testing*. Artech House Computer Library, 2003.

Dustin, Elfriede. *Automated Software Testing: Introduction*, Management, and Performance. Addison-Wesley, 1999.

Dustin, Elfriede, et al. *Quality Web Systems: Performance, Security, and Usability.* Addison-Wesley, First Edition, 2001.

Hayes, Linda G. *Automated Testing Handbook*. Software Testing Institute, 1996.

Hetzel, Bill. *The Complete Guide to Software Testing*. John Wiley & Sons, Inc., Second Edition, 1993.

Jorgensen, Paul C. *Software Testing: A Craftsman's Approach.* CRC Press, 1995.

Kaner, Cem, et al. *Lessons Learned in Software Testing*. John Wiley & Sons, Inc., First Edition, 2001.

Kit, Edward. *Software Testing in the Real World*. Addison-Wesley, First Edition, 1995.

Lewis, William E. *Software Testing and Continuous Quality Improvement*. Auerbach Publishers, Second Edition, 2000.

Li, Kanglin. *Effective Software Test Automation: Developing an Automated Software Testing Too*l. Sybex Inc., First Edition, 2004.

Marick, Brian. *Craft of Software Testing: Subsystems Testing Including Object-Based and Object-Oriented Testing*. Prentice Hall, 1994.

Marshall, Steve, et al. *Making E-Business Work: A Guide to Software Testing in the Internet Age*. Newport Press Publications, 2000.

Mosley, Daniel J. *Client Server Software Testing on the Desktop and the Web*. Prentice Hall, First Edition, 1999.

Mosley, Daniel J. and Bruce A. Posey. *Just Enough Software Test Automation.* Prentice Hall, First Edition, 2002.

Nguyen, Hung Q. *Testing Applications on the Web: Test Planning for Internet-Based Systems*. John Wiley & Sons, Inc., First Edition, 2000.

Patton, Ron. *Software Testing*. Sams, 2000.

Perry, William E. *Effective Methods for Software Testing*, John Wiley & Sons, Inc., 2000.

Perry, William E. and Randall W. Rice. *Surviving the Top Ten Challenges of Software Testing: A People-Oriented Approach*. Dorset House Publishing Company, Inc., 1997.

Pham, Hoang. *Software Reliability and Testing*. IEEE Computer Society Press, First Edition, 1995.

Poston, Robert M. *Automating Specification-Based Software Testing.* Institute of Electrical & Electronics Engineers, Inc., 1996.

Siegel, Shel. *Object-Oriented Software Testing: A Hierarchical Approach*. John Wiley & Sons, Inc., First Edition, 1996.

Sykes, David A. and John D. McGregor. *Practical Guide to Testing Object-Oriented Software*. Addison-Wesley, First Edition, 2001.

Appendix

C

How to Take the CSTE Examination

T the Introduction of this preparation guide explained a process for you to follow to prepare for the examination. It emphasized familiarizing yourself with a Common Body of Knowledge (CBOK), the vocabulary of software quality, the activities performed by testing professionals, and reviewing several other different references that are included on the software certifications Web site. Other references you might look at include articles in professional IT publications.

If you feel you are ready to take the examination, you need to schedule the examination. Be sure to visit www.softwarecertifications.org for up-to-date CSTE examination dates and places. Once scheduled, the remaining event is to take the examination.

CSTE Examination Overview	**C-1**
Guidelines to Answer Questions	**C-2**
Sample CSTE Examination	**C-5**

C.1 CSTE Examination Overview

The four and a half hour examination consists of four written parts, including multiple-choice and essay questions. A typical CSTE examination is comprised of two parts for Software Testing Theory and two parts for Software Testing Practice:

C.1.1 Software Testing Theory

Part 1 50 multiple-choice questions. Complete within 45 minutes.

Part 2 10 essay questions. Complete within 1 hour, 15 minutes.

Software testing theory evaluates your understanding of testing principles, practices, vocabulary and concepts. In other words, do you have a solid foundation in testing basics? For example, can you differentiate between verification and validation?

C.1.2 Software Testing Practice

Part 3 50 multiple-choice questions. Complete within 45 minutes.

Part 4 10 essay questions. Complete within 1 hour, 15 minutes.

Software testing practice evaluates whether you can apply testing basics to real-world situations. For example, a question may be: "What methods of quality control would you use to reduce defects for a software system under development? During which phase of development would you use those methods?"

You cannot bring any study or supporting materials to the examination site other than a pencil. Each individual will have an assigned seat. You may take a 15-minute break between each examination part, but not during the examination.

Proctors for the examination are not required to be certified. If not certified, they are ineligible to take the examination for at least two years after proctoring. Proctors follow specific instructions to conduct the examination. Software Certifications' policies and procedures enforce confidentiality and security of the examination instrument, prior to and after the examination.

C.2 Guidelines to Answer Questions

The examination proctor will give you one part of the examination at a time. As you receive each part, use the following steps to answer the questions:

1. Read the entire examination part before answering any questions.
   - For multiple-choice parts, only read each question's stem, not the four to five responses.
   - For essay parts, read all of the essay questions thoroughly.
2. As you read through each question (multiple-choice and essay) determine whether:
   - You absolutely know the answer to this question.
   - You believe you know the answer to this question.

- You are not sure you know the answer, or it would take time to develop an answer.

3. For both multiple-choice and essay questions, answer the questions that you know the answers; they should not take you much time to complete. This will give you the majority of the examination time for the other questions you may need more time to answer.

4. Answer the questions that you believe you know the answer.
   - For multiple-choice questions, answer all of the questions.
   - For essay questions, answer the questions worth the most points first, followed by those worth less points. Note that the points equal the percentage of score allocated to that essay question.

5. Answer the questions that you do not know the answer.
   - For multiple-choice questions, answer all of the questions.
   - For essay questions, answer the questions worth the most points first, followed by those worth less points.

Follow these recommended guidelines for **answering essay questions**. Remember that an individual grades your examination. Make your thoughts and ideas clear and concise. Also, be sure to write legibly.

- Those questions worth the most points should be the longest essay response. For example, if an essay question is worth 25 points, the response should be at least twice as long as an essay response for a question worth 10 points.
- You need not write complete sentences. Those grading your examination are looking for key phrases and concepts. You can highlight them in a bulleted form, underlined or capitalized. This will help ensure that the individual scoring your examination can readily understand your knowledge of the correct response to that essay question.
- Charts, graphs, and examples enhance your responses and enable the individual grading your examination to evaluate your understanding of the question. For example, use a control chart example to clearly indicate your knowledge of statistical process control.

Follow these recommended guidelines for **answering multiple-choice questions**.

- Each multiple-choice question is comprised of a stem statement (or stem) and multiple responses to the stem. Read the stem carefully to assure you understand what is being asked. Then without reading the given responses to the stem, create in your mind what you believe would be the correct response. Look for that response in the list of responses.
- You will be given four or five responses to each stem.
- If you cannot create a response to the stem prior to reading the responses, attempt to eliminate those responses which are obviously wrong. In most cases, after that you will only have two responses remaining.
- To select between what appears to be correct responses, rethink the subject matter in the stem and document what you know about that subject. This type of analysis should

help you to eliminate a response or select the correct response among what should be two responses.

Follow these recommended guidelines when you **do not know the answer to a question**. Usually you can take a few minutes to look over the choices or write down some key points to help you answer the questions.

- For multiple-choice questions, answer all questions – there is no reduction to your score for wrong answers. If you do not know the answer, try to rule out some of the potential responses. Then select the response you believe might be correct.
- For essay questions, indicate any information that might be helpful. For example, you have a question on test policies. You have not actually written a test policy, but you could define a test policy, give an example of a test policy, and explain what you recommend to implement that policy in your IT organization.

C.3 Sample CSTE Examination

The following CSTE examination is a sample of some of the questions, both multiple-choice and essay, which may appear on the actual examination. Use this sample examination to help you study. The multiple-choice questions are listed by skill category. If you miss a question in a particular skill category, simply refer back to that skill category in this guide for additional study. Use the essay question responses in the sample examination as a guide for responding to the actual CSTE essay questions.

C.3.1 Part 1 and Part 3 Multiple-Choice Questions

This sample examination contains 20 multiple-choice questions – two questions for each category. While taking this sample examination, follow the steps described in "Guidelines to Answer Questions" on page C-2 to practice the recommended techniques.

Circle the correct answer. Compare your answers to the answer key that follows on page 11.

Skill Category 1 – Software Testing Principles and Concepts

1. The customer's view of quality means:

    a. Meeting requirements

    b. Doing it the right way

    c. Doing it right the first time

    d. Fit for use

    e. Doing it on time

2. The testing of a single program, or function, usually performed by the developer is called:

a. Unit testing

b. Integration testing

c. System testing

d. Regression testing

e. Acceptance testing

Skill Category 2 – Building the Test Environment

3. The measure used to evaluate the correctness of a product is called the product:
   a.Policy

   b.Standard

   c.Procedure to do work

   d.Procedure to check work

   e.Guideline

4. Which of the four components of the test environment is considered to be the most important component of the test environment:
   a.Management support

   b.Tester competency

   c.Test work processes

   d.Testing techniques and tools

Skill Category 3 – Managing the Test Project

5. Effective test managers are effective listeners. The type of listening in which the tester is performing an analysis of what the speaker is saying is called:
   a.Discriminative listening

   b.Comprehensive listening

   c.Therapeutic listening

   d.Critical listening

   e.Appreciative listening

6. To become a CSTE, an individual has a responsibility to accept the standards of conduct defined by the certification board. These standards of conduct are called:
   a.Code of ethics

   b.Continuing professional education requirement

   c.Obtaining references to support experience

   d.Joining a professional testing chapter

   e.Following the common body of knowledge in the practice of software testing

Skill Category 4 – Test Planning

7. Which of the following are risks that testers face in performing their test activities:
   a.Not enough training

   b.Lack of test tools

   c.Not enough time for testing

   d.Rapid change

   e.All of the above

8. All of the following are methods to minimize loss due to risk. Which one is *not* a method to minimize loss due to risk:
   a.Reduce opportunity for error

   b.Identify error prior to loss

   c.Quantify loss

   d.Minimize loss

   e.Recover loss

Skill Category 5 – Executing the Test Plan

9. Defect prevention involves which of the following steps:
   a.Identify critical tasks

   b.Estimate expected impact

   c.Minimize expected impact

   d.a, b and c

   e.a and b

10. The first step in designing use case is to:
    a.Build a system boundary diagram

    b.Define acceptance criteria

    c.Define use cases

    d.Involve users

    e.Develop use cases

Skill Category 6 – Test Reporting Process

11. The defect attribute that would help management determine the importance of the defect is called:
    a.Defect type

    b.Defect severity

    c.Defect name

    d.Defect location

    e.Phase in which defect occurred

12. The system test report is normally written at what point in software development:
    a.After unit testing

    b.After integration testing

    c.After system testing

    d.After acceptance testing

Skill Category 7 – User Acceptance Testing

13. The primary objective of user acceptance testing is to:
    a.Identify requirements defects

    b.Identify missing requirements

    c.Determine if software is fit for use

    d.Validate the correctness of interfaces to other software systems

    e.Verify that software is maintainable

14. If IT establishes a measurement team to create measures and metrics to be used in status reporting, that team should include individuals who have:
    a.A working knowledge of measures

    b.Knowledge in the implementation of statistical process control tools

    c.A working understanding of benchmarking techniques

    d.Knowledge of the organization's goals and objectives

    e.All of the above

Skill Category 8 – Testing Software Developed by Contractors

15. What is the difference between testing software developed by a contractor outside your country, versus testing software developed by a contractor within your country:
    a.Does not meet people needs

    b.Cultural differences

    c.Loss of control over reallocation of resources

    d.Relinquishment of control

    e.Contains extra features not specified

16. What is the definition of a critical success factor:
    a.A specified requirement

    b.A software quality factor

    c.Factors that must be present

    d.A software metric

    e.A high cost to implement requirement

Skill Category 9 – Testing Internal Control

17. The condition that represents a potential for loss to an organization is called:
    a.Risk

    b.Exposure

    c.Threat

    d.Control

    e.Vulnerability

18. A flaw in a software system that may be exploited by an individual for his or her advantage is called:
    a.Risk

    b.Risk analysis

    c.Threat

    d.Vulnerability

    e.Control

Skill Category 10 – Testing New Technologies

19. The conduct of business of the Internet is called:
    a.e-commerce

    b. e-business

    c. Wireless applications

    d. Client-server system

    e. Web-based applications

20. The following is described as one of the five levels of maturing a new technology into an IT organization's work processes. The "People-dependent technology" level is equivalent to what level in SEI's compatibility maturity model:
    a.Level 1

    b.Level 2

    c.Level 3

    d.Level 4

    e.Level 5

C.3.2 Part 1 and Part 3 Multiple-Choice Answers

The answers to the sample examination for Part 1 and Part 3 are as follows. If you missed a question, study that material in the relative skill category.

1. d Fit for use
2. a Unit testing
3. b Standard
4. a Management support
5. d Critical listening
6. a Code of ethics
7. e All of the above
8. c Quantify loss
9. d a, b and c
10. a Build a system boundary diagram
11. b Defect severity
12. c After system testing
13. c Determine if software is fit for use
14. e All of the above
15. b Cultural differences
16. c Factors that must be present
17. a Risk
18. d Vulnerability
19. b e-business
20. a Level 1

C.3.3 Part 2 and Part 4 Essay Questions and Answers

Essay questions on theory focus on "what to do" and essay questions on best practices focus on "how to do it." Part 2 and Part 4 of the certification exam both have ten essay questions, one from each skill category. Five of the following essay questions are questions that could be included in Part 2, Theory Essay Questions of the CSTE examination. The other five questions are questions that could be included in Part 4, Practice Essay Questions.

C.3.3.1Part 2 – Software Testing Theory Essay Questions

Answer these five essay questions following the guidelines in Guidelines to Answer Questions on page C-2. Note that on the actual examination, each page has just one essay question to give you plenty of space to write your response.

Skill Category 1 – Software Testing Principles and Concepts

1. List 5 reasons why we test software.

Skill Category 5 – Executing the Test Plan

2. The requirements for a major system to which you have been assigned as a tester includes a very complex data model with an extensive list of fields, codes, and data values that are highly interdependent. What steps will you take throughout your test planning to assure the completeness and adequacy of your test coverage and what impact will these steps have on the content of your test plans?

Skill Category 2 – Building the Test Environment

3. Your IT Director is concerned about the cost of software testing and the length of time involved in testing a software system. An investigation indicates that a significant amount of testing is performed after it is known that conditions have occurred which makes the additional testing of little value. You believe that if testers stop testing when certain conditions are met, the cost and duration of software testing could be reduced. Your IT Director agrees with you and asks you to indicate what types of conditions you would include in a test plan that would justify stopping software testing. These conditions would have been corrected prior to resuming the tests.

   List below those conditions that you would recommend being incorporated into a test plan for testers to stop testing. Name and briefly describe each condition.

Skill Category 8 – Testing Software Developed by Contractors

4. Your organization has outsourced the development of a major software project. Your manager has asked you to develop a plan to test the software; but, before developing your test plan your manager wants you to list at least four differences between testing software developed by contractors and software developed in-house. List these differences:

Skill Category 3 – Managing the Test Project

5. Developing compatibility and motivation with a test team helps assure effective testing. List below at least four guidelines you would follow to develop compatibility and motivation within your test team.

C.3.3.2Part 2 – Software Testing Theory Essay Answers

The following responses are examples of responses expected to receive a good grade. Review these examples as responses that adequately answer the essay question, not as the only correct response.

Essay 1.

a. *To produce a quality product, which is defect free, and also to make sure all the requirements are satisfied and the best design system architecture is used.*

b. *Customers/user satisfaction (customers are the king).*

c. *To make sure the software is:*

Correct

Reliable

Portable

Efficient

Interoperable

Usable

Maintainable

Re-usable

Secure

Flexible

d. *To achieve the goals of an organization as well as to make profit.*

e. *To reduce the possible risks associated with the software, then reduce the loss, which might happen when/if the software is released with defects included.*

Essay 2.

a.Since it is a very complex model and if I have access to source code, then try to test the code with the following techniques:

Branch coverage

Statement coverage

Decision coverage

b.Would use Boundary Value Analysis, error guessing and Equivalence partition techniques to create 3 or more test cases for every field, in addition to the test cases that are created for testing functional requirements.

c.Would test the control mechanism implemented for every field and make sure the fields accept only the values that they are supposed to accept. For example, will make sure alphabetic fields don't accept numbers or special characters.

d.Check for lengths to make sure the input data cannot exceed the field length defined in the database.

e.Would test for valid dates, if date fields were used.

f.Would create a data pool with all possible values for fields and generate quite a few test scripts to test the different fields at a faster speed.

g.Would make sure all possible error conditions are tripped and they are handled gracefully.

h.Would test all the required fields as mandatory.

i.Would alter the auto-generating fields in the database (if possible) and see how the system handles it.

Essay 3.

a.When the quality objectives are met.

b.When the exit criteria mentioned in the test plan is met.

c.When there is no potential show-stopper or show blocker in the software.

d.When the expected test coverage is achieved.

e.When all the test cases are executed.

f.When the project budget is depleted or when test time is not enough (this is not a good practice, but happens.)

g.When the remaining minor defects go below the accepted level.

Essay 4.

The differences between software developed by a contractor and software developed in-house are:

- *Quality factors may not be specified.*
- *There are many factors such as reliability and ease of use which are frequently not included as part of the contractual criteria. Thus when the software is delivered it may not be as easy to use or as reliable as desired by the contractor.*
- *Non-testable requirements and criteria.*
- *If the requirements or contractual criteria in measurable and testable terms then the delivered result may not meet the intent of the contractor.*
- *Customer's standards may not be met*
- *Unless the contract specifies the operational standards and documentation standards the delivered product may be more complex to use than desired by the customer.*
- *Missing requirements*
- *Unless detailed analysis and contractual specifications work is complete the contractor may realize during the development of the software that requirements are missing and thus the cost of the contract could escalate significantly.*
- *Overlooked changes in standards in technology*
- *If changes in standards that the organization must meet, or the introduction of new desirable technology is incorporated into the contract there may be significant cost to modify the software for those new standards in technology.*
- *Training and deployment may be difficult*
- *If software is developed by another organization there may be inadequate knowledge in the contracted organization to provide the appropriate training for staff and to ensure that deployment is effective and efficient.*

Essay 5.

Guidelines are helpful in developing compatibility and motivation of a software project team:

2.Communicate the vision, objectives, and goals of the project.

A software professional wants to know what the project is trying to accomplish. The vision indicates why the project is undertaken, the goals and objectives indicate what the project is to achieve. For example, the vision of a bank commercial loan software project might be to increase profitability. This specific objective might be to provide the loan officer the information needed to make a good loan decision.

3.Define roles and responsibilities of team members.

Software projects, unlike non-software projects, have roles which are heavily people dependent and project scope dependent. It's important for professional staff to have

those roles and responsibilities clearly defined. The staffing matrix defines those roles and responsibilities.

4.Empower team members to manage their responsibilities.

Empowerment is a major motivator for professional people. Many of the agile concepts relate to empowerment. In other words, enable people to perform the tasks in the most efficient and effective manner. This helps eliminate barriers that increase costs and help project schedule.

5.Hold team members accountable for their assigned responsibilities in the team process.

Team members need to have their work tasks well defined and then held accountable for completing those work tasks. Managerial practices indicate that this process works best when individuals accept responsibility for performing tasks. Thus, having the Project Manager work individually with team members to assign team tasks they agree to perform, and then hold those individuals accountable for completing those tasks is an effective managerial practice.

6.Ensure that all the required skills are present on the team.

Projects cannot be completed successfully if the team members lack the skills to complete the project. It is not necessary for every team member to have all the needed skills, but the team in total needs the skills. The staffing matrix helps assure that the appropriate skills exist within the project team.

7.Provide the necessary technical and team training.

If the team lacks technical and team skills, the project manager should provide that training. Technical skills include the skills necessary to design and build the software, team skills to cover such skills as consensus building and conflict resolution.

8.Award successes and celebrate achievements.

Establishing goals and objectives provides the basis for rewards and celebrations. While it's appropriate to reward and celebrate individual achievements, the team building necessitates team goals and team celebrations. These can be centered around milestones accomplished, as well as scoring high on customer satisfaction surveys.

C.3.3.3Part 4 – Software Testing Practice Essay Questions

Answer these five essay questions following the guidelines in Guidelines to Answer Questions on page C-2. Note that on the actual examination, each page has just one essay question to give you plenty of space to write your response.

Skill Category 1 – Software Testing Principles and Concepts

1. Explain and give an example of each of the following black-box test case techniques.
   Equivalence partitioning:

   Boundary analysis:

   Error guessing:

Skill Category 1 – Software Testing Principles and Concepts

Explain the difference between verification and validation.

Skill Category 4 – Test Planning

2. A study by a major software supplier indicates that software testers make more defects than software developers. Since software testing in your organization is a costly and time-consuming process, your IT Director believes that software testers may, in fact, make more defects than software developers. The IT Director has asked you to describe what you believe might be the five major defects that software testers make in planning, executing and reporting the results from software testing.

   List below the name you would give to those five defects and briefly describe each defect. (10 points)

Skill Category 1 – Software Testing Principles and Concepts

3. Assume you have been promoting testing throughout the development life cycle, but your manager does not really understand what specifically is involved. Your manager has asked you to present the concept at an IT staff meeting. You chose the "V" concept of testing model to explain testing throughout the development life cycle.

   Explain and provide a graphic below of the "V" concept of testing.

Skill Category 5 – Executing the Test Plan

4. Recording and tracking defects uncovered during testing is an important test responsibility. Describe three attributes of a defect that you believe testers should include when they document a defect. Give specific examples of how you would describe each of the three defect attributes you propose.

C.3.3.4Part 4 – Quality Assurance Practice Essay Answers

The following responses are examples of responses expected to receive a good grade. Review these examples as responses that adequately answer the essay question, not as the only correct response.

Essay 1.

Equivalence partitioning:

This technique will help to narrow down the possible test cases using equivalence classes. Equivalence class is one, which accepts same types of input data. Few test cases for every equivalence class will help to avoid exhaustive testing.

Boundary analysis:

This technique helps to create test cases around the boundaries of the valid data. Usually values passed are exact boundary values, + or – 1 at the lower boundary and + or – 1 at the higher boundary. This is an excellent technique and has proven that software is error-proof to boundaries.

Error guessing:

This technique is used to find defects using the experience of the tester, who is very familiar to the module he/she is testing. Usually the history is used as an input to guess the values for input. For example, if the software is always error prone to negative values, where only positive values should be accepted, the tester can easily guess to enter negative values for various tests that would help to identify all the defects related to them.

Essay 2.

Verification:

- *Uses non-executable methods of analyzing the various artifacts.*
- *More effective, it has been proven that 65% of defects can be discovered here.*
- *Uses inspection reviews to verify the requirements and design.*
- *Good examples are having checksheets, traceability matrix kind of documents to verify the documents, requirements and software features.*
- *Code reviews, walkthroughs also come under this category.*

Validation:

- *Can be used throughout the software development life cycle.*
- *Uses executable methods – means the software will be used to analyze the various artifacts and test software.*
- *Effective, but not as effective as verification, for removing defects. It has been proven that 30% of defects can be discovered here.*
- *Software will be executed to validate the requirements and design features.*
- *Using functional or structural testing techniques to catch defects.*
- *Unit testing, coverage analysis, black-box techniques fall under this category.*
- *It can also be used throughout the life cycle.*

Essay 3.

Inadequate test requirements: test requirements are created from functional requirements and the functional requirements are not good and complex enough.

Testers are in lose-lose situation: if we find more defects we get blamed for slower project implementation, and if we find less defects, the quality of our team is in jeopardy.

Dependence on independent testers: unit testing is not good enough and this causes testers to test and find all defects, which puts more pressure and workload on them.

Inadequate test coverage: because of frequent software developments, testers did not get a chance to test all test cases at once; this causes too much duplication of work.

Test plan updates not enough: test cases are not added because of time crunch and this caused some test cases to ship during regression.

Changes not tested: changes made in the code were not tested, resulting in shipping defects to operations.

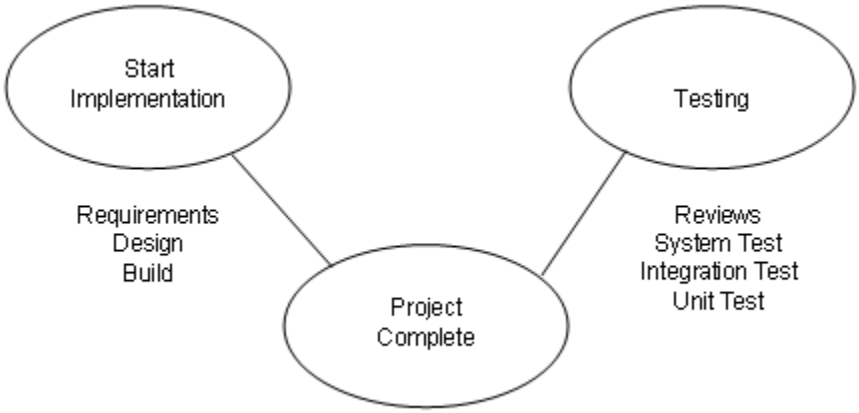
Essay 4.

Life cycle testing involves continuous testing of the system during the developmental process. At predetermined points, the results of the development process are inspected to determine the correctness of the implementation. These inspections identify defects at the earliest possible point.

Life cycle testing cannot occur until a formalized SDLC has been incorporated. Life cycle testing is dependent upon the completion of predetermined deliverables at specified points in the developmental life cycle. If information services personnel have the discretion to determine the order in which deliverables are developed, the life cycle test process becomes ineffective. This is due to variability in the process, which normally increases cost.

The life cycle testing concept can best be accomplished by the formation of a test team. The team is comprised of members of the project who may be both implementing and testing the system. When members of the team are testing the system, they must use a formal testing methodology to clearly distinguish the implementation mode from the test mode. They also must follow a structured methodology when approaching testing the same as when approaching system development. Without a specific structured test methodology, the test team concept is ineffective because team members would follow the same methodology for testing as they used for developing the system. Experience shows people are blind to their own mistakes, so the effectiveness of the test team is dependent upon developing the system under one methodology and testing it under another.

The life cycle testing concept is illustrated below This illustration shows that when the project starts both the system development process and system test process begins. The team that is developing the system begins the systems development process and the team that is conducting the system test begins planning the system test process. Both teams start at the same point using the same information. The systems development team has the responsibility to define and document the requirements for developmental purposes. The test team will likewise use those same requirements, but for the purpose of testing the system. At appropriate points during the developmental process, the test team will test the developmental process in an attempt to uncover defects. The test team should use the structured testing techniques outlined in this book as a basis of evaluating the system development process deliverables.



Essay 5.

The three defect attributes I would propose are:

- *Defect Naming*
  Name defects according to the phase in which the defect most likely occurred, such as a requirements defect, design defect, documentation defect, and so forth.

- *Defect Severity*
  Use three categories of severity as follows:

  - *Critical – Would stop the software system from operating.*

  - *Major – Would cause incorrect output to be produced.*

  - *Minor – Would be a problem, but would not cause improper output to be produced, such as a system documentation error.*

- *Defect Type*
  Use the following three categories:

  - *Missing – A specification not included in the software.*

  - *Wrong – A specification improperly implemented in the software.*

  - *Extra – Element in the software not requested by a specification.*

This page intentionally left blank.