CHAPTER 01

THE SCOPE OF SOFTWARE ENGINEERING

Topics

- Introduction
- Historical Aspects
- Economic Aspects
- Maintenance Aspects
- Requirements, Analysis, and Design Aspects
- Team Development Aspects
- Why There is No Planning Phase
- Why There is No Testing Phase
- Why There is No Documentation Phase
- The Object-Oriented Paradigm
- The Object-Oriented Paradigm in Perspective
- Terminology
- Ethical Issues

Reference

- Schach, S.R. (2010). *Object-Oriented and Classical Software Engineering*, Eighth Edition. McGraw-Hill, ISBN: 978-0-07-337618-9.

Introduction

- Software engineering is a discipline whose aim is the production of fault-free software, delivered on time and within budget, that satisfies the user's needs. Furthermore, the software must be easy to modify when the user's needs change.
- Scope of software engineering is extremely broad
- Examine different aspects

Historical Aspects

- A NATO study group in 1967 coined the term software engineering
  - Building software is similar to other engineering tasks
- Software engineering should use the philosophies and paradigms of established engineering disciplines to solve the software crisis
  - Quality of software generally was unacceptably low
  - Deadlines and budgets were not being met
- Despite many success stories, a large proportion of software products still are being delivered late, over budget, and with residual faults.
- Studies by Standish Group
  - Research firm that analyzes software development projects

Historical Aspects

- Study of 280,000 development projects completed in 2000
  - Only 28% of the projects were successfully completed
- Study of 9,236 development projects completed in 2004
  - Only 29% of the projects were successfully completed
- Study completed in 2006
  - Only 35% of the projects were successfully completed
- Figure 1.1

Historical Aspects

- Survey conducted by the Cutter Consortium in 2002
  - Astounding 78% of information technology organizations involved in disputes that ended in litigation
  - In 67% of those cases, the functionality or performance of the software product did not meet up to the claims of the software developers
  - In 56% of those cases, the promised delivery date slipped several times
  - In 45% of those cases, the faults were so severe that the software product was unusable
- A software engineer has to acquire a broad range of skills
  - Both technical and managerial skills
  - Skills applied to every step of software production

Economic Aspects

- Software engineer is interested in techniques that make sound economic sense
  - The cost of introducing new technology into an organization
  - The cost of maintenance

Maintenance Aspects

- Maintenance is described within the context of the software life cycle
- Life-cycle model — Description of the steps performed when building a software product
  - Many different models
- Phases — Life-cycle model broken into a series of smaller steps
  - Easier to perform a sequence of smaller tasks
  - Number of phases varies from model to model
- Life cycle of a product — Actual series of steps performed on the software product
  - From concept exploration through final retirement
  - In Contrast to the life-cycle model (a theoretical description)
  - Phases of the life cycle may not be carried out as specified

Maintenance Aspects

- Although there are many variations, a software product goes through six phases

(1) Requirements phase
  - Concept is explored and refined
  - Client's requirements are elicited

(2) Analysis (specifications) phase
  - Client's requirements are analyzed and presented in the form of the specification document ("what the product is supposed to do")
  - Software Project Management Plan is drawn up, describing the proposed development in detail

(3) Design phase
  - Two consecutive processes of architectural design (product is broken down into modules) and detailed design (each module is designed)

Maintenance Aspects

  - Resulting in two design documents ("how the product does it")

(4) Implementation phase
  - Various components undergo coding and testing (unit testing)
  - The components are combined and tested as a whole (integration)
  - When the developers are satisfied that the product functions correctly, it is tested by the client (acceptance testing)
  - Ends when product accepted by client and installed on client's computer

(5) Postdelivery maintenance
  - All changes to product after delivery
  - Includes corrective maintenance (software repair): removal of residual faults while leaving the specifications unchanged and enhancements (software updates): changes to the specifications and the implementation of those changes

Maintenance Aspects

  - Two types of enhancements are perfective (changes the client thinks will improve the effectiveness of the product, such as additional functionality or decreased response time) and adaptive (changes made in response to changes in the environment, such as new hardware/operating system or new government regulations)

(6) Retirement
  - Product removed from service
  - Functionality provided no longer of use to client

Classical and Modern Views of Maintenance

- In 1970s, software production viewed as two distinct activities of development followed by maintenance
  - Described as the development-then-maintenance model
  - A temporal definition (an activity is classified depending on when it is performed)

Maintenance Aspects

- The development-then-maintenance model is unrealistic today
  - Construction of a product can take a year or more, during which the client's requirements may change
  - Developers have to perform maintenance before product is installed
  - Developers try to reuse parts of existing software products
- More realistic: Maintenance is the process that occurs when software undergoes modifications to code and associated documentation due to a problem or the need for improvement or adaptation [ISO/IEC 12207]
  - An operational definition: Irrespective of when the activity takes place
  - Definition adopted by [IEEE/EIA 12207.0]
  - Postdelivery maintenance is a subset of modern maintenance

Maintenance Aspects

- ISO — International Organization for Standardization
  - A network of national standard institutes
  - 147 countries
  - Central secretariat based in Geneva, Switzerland
  - Published over 13,500 internationally accepted standards
- IEC — International Electrotechnical Commission
  - A non-profit, non-governmental organization
  - Prepares and publishes international standards for all electrical, electronic and related technologies
  - Manages conformity assessment systems that certify that equipment, systems or components conform to standards
- EIA — Electronic Industries Alliance
  - A trade organization
  - Composed as an alliance of trade associations
  - For electronics manufacturers in the United States

Maintenance Aspects

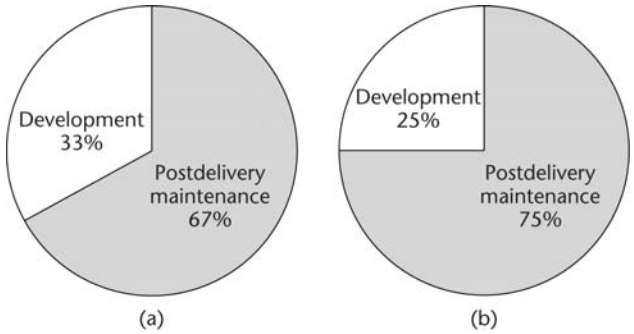
- IEEE — Institute of Electrical and Electronics Engineers
  - An international non-profit, professional organization
  - For the advancement of technology related to electricity
  - World's largest professional association
  - More than 395,000 members in around 150 countries
  - Highly cited publications, conferences, technology standards, and professional and educational activities

The Importance of Postdelivery Maintenance

- False: Bad software products undergo postdelivery maintenance
  - Bad software products are thrown away
  - Good software products are repaired and enhanced
- A software product is a model of the real world
  - Real world is perpetually changing
  - Software has to be maintained constantly

Maintenance Aspects

- How much time (money) is devoted to postdelivery maintenance
  - Some 40 years ago, approximately two-thirds of total software costs
  - Newer data show a larger proportion (70% to 80%)
- Figure 1.3



Maintenance Aspects

- Average cost percentages of the classical development phases have not changed much
- Figure 1.4

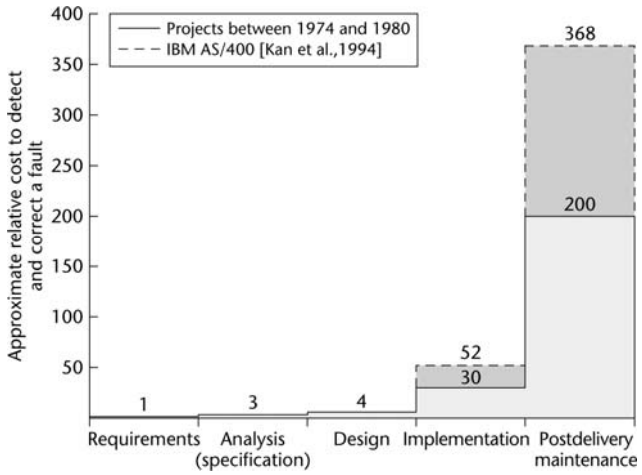
	Various Projects between 1976 and 1981	132 More Recent Hewlett-Packard Projects
Requirements and analysis (specification) phases	21%	18%
Design phase	18	19
Implementation phase		
Coding (including unit testing)	36	34
Integration	24	29

Requirements, Analysis, and Design Aspects

- Cost of correcting a fault increases steeply throughout the phases
  - The earlier one corrects a fault, the better
  - A fault in requirements may also appear in the specifications, the design, and the code
  - Studies have shown that between 60% and 70% of all faults detected in large projects are requirements, analysis, or design faults
  - It is important to improve requirements, analysis, and design techniques

Requirements, Analysis, and Design Aspects

- Figure 1.6



Team Development Aspects

- Most software is produced by a team of software engineers
  - Team development leads to interface problems among code components and communication problems among team members
  - Unless the team is properly organized, an inordinate amount of time can be wasted in conferences between team members
  - The scope of software engineering must include techniques for ensuring that teams are properly organized and managed
- Scope of software engineering is extremely broad
  - Every step of the software life cycle
  - Human aspects such as team organization
  - Economic aspects
  - Legal aspects such as copyright laws

Why There is No Planning Phase

- Impossible to develop a software product without a plan
  - Essential to have a planning phase at the beginning of project
  - Until it is known exactly what is to be developed, an accurate detailed plan cannot be drawn up
- Three types of planning activities take place

  (1) At the beginning of the project, preliminary planning takes place for managing the requirements and analysis phases

  (2) Once what is going to be developed is known precisely, the software project management plan (SPMP) is drawn up, which includes the budget, staffing requirements, and detailed schedule

  Earliest when specification document is approved by the client

  (3) Through the project, management monitors the SPMP (watch for any deviation)
- There is no separate planning phase
  - Planning activities are carried out all through the life cycle

Why There is No Testing Phase

- Checking the software product once it is ready to be delivered is too late
- Although there are times when testing predominates, there should never be times when no testing is being performed
- Testing predominates
  - Toward the end of each phase (verification) and
  - Before the product is handed over to the client (validation)
- If testing is treated as a separate phase
  - Danger that testing will not be carried out constantly throughout
- Every software development organization should contain an independent group, called the software quality assurance (SQA) group, whose primary responsibility is to ensure that
  - Product is what the client needs
  - Product has been built correctly
- Quality — Extent to which software meets its specifications

Why There is No Documentation Phase

- At all times, the documentation of a software product must be complete, correct, and up to date
  - Large turnover in personnel in the software industry
- Impossible to perform the steps of a specific phase without the documentation of the previous phase
- Impossible to test a software product without documents that state how the product is supposed to behave
- Impossible to perform maintenance without documentation that describes precisely what the product does

The Object-Oriented Paradigm

- Prior to 1975: No specific techniques
- 1975 to 1985: Structured or classical paradigm
- Unable to cope with the increasing size of software products
  - Adequate for small-scale (5,000 Lines of Code) and medium-scale (50,000 LOC) products
  - Could not scale up to large-scale (500,000 LOC)
  - Products with 5 million lines of code are not unusual
- Did not live up to the expectations during postdelivery maintenance
  - Two-thirds of the budget for postdelivery maintenance
  - Classical paradigm did not change that
- Either operation (action) oriented or attribute (data) oriented
  - Not both
  - Basic components of a software product: Operations of the product and attributes on which those operations operate

The Object-Oriented Paradigm

- Object-Oriented Paradigm
  - Considers both attributes and operations to be equally important
  - Object — A unified software artifact that incorporates both the attributes and the operations performed on the attributes
- Artifact — A component of a software product
  - E.g., a specification document, a code module, or a manual
- Strengths of the object-oriented paradigm

  (1) Makes maintenance quicker and easier, and the chance of introducing a regression fault (a fault inadvertently introduced as a consequence of a change) is greatly reduced

  (2) Makes development easier: The close correspondence between the objects and their counterparts in the real world

  (3) Well-designed objects are independent units: The conceptual independence is termed encapsulation and information hiding ensures the implementation details are hidden

The Object-Oriented Paradigm

(4) The product consists of a number of smaller, largely independent units: Reduces the complexity of a software product
A product in the classical paradigm is implemented as a set of modules, but it is essentially a single unit: Less successful with larger products

(5) Promotes reuse: Objects are independent entities and can be utilized in future products

- Object-oriented paradigm has a modified life cycle model
  - Phase for classical paradigm versus workflow for object-oriented
  - Terms are distinct
- One of the steps of object-oriented analysis work flow is to determine classes
  - Because class is a kind of module, architectural design is performed during the object-oriented analysis work flow

The Object-Oriented Paradigm

- In classical paradigm, a sharp transition between the analysis phase and the design phase
  - From "what to do" to "how to do it"
- In object-oriented analysis, objects enter the life cycle from the very beginning
  - Objects are extracted in the analysis workflow
  - Designed in the design workflow
  - Coded in the implementation workflow
- Object-oriented paradigm an integrated approach with smooth transitions

The Object-Oriented Paradigm

- Figure 1.8 and Figure 1.9

Classical Paradigm	Object-Oriented Paradigm
1. Requirements phase	1. Requirements workflow
2. Analysis (specification) phase	2′. Object-oriented analysis workflow
3. Design phase	3′. Object-oriented design workflow
4. Implementation phase	4′. Object-oriented implementation workflow
5. Postdelivery maintenance	5. Postdelivery maintenance
6. Retirement	6. Retirement

Classical Paradigm	Object-Oriented Paradigm
2. Analysis (specification) phase • Determine what the product is to do	2′. Object-oriented analysis workflow • Determine what the product is to do • Extract the classes
3. Design phase • Architectural design (extract the modules) • Detailed design	3′. Object-oriented design workflow • Detailed design
4. Implementation phase • Code the modules in an appropriate programming language • Integrate	4′. Object-oriented implementation workflow • Code the classes in an appropriate object-oriented programming language • Integrate

The Object-Oriented Paradigm in Perspective

- The issues with object-oriented paradigm
  - The object-oriented paradigm has to be used correctly (just as easy to misuse as any other paradigm)
  - When correctly applied, object-oriented paradigm can solve some (but not all) of the problems of classical paradigm
  - The object-oriented paradigm has some problems of its own (discussed later)
  - The object-oriented paradigm is the best approach available today, and it can be superseded by a superior technology in the future

Terminology

- Client — The individual who wants a product to be built (developed)
- Developers — Members of team responsible for building the product
- Internal software development — Both the client and developers are part of the same organization
- Contract software — Client and the developers are independent organizations
- User — The person) on whose behalf the client has commissioned the product and who will utilize the software
- Custom software — Developed for one client
- Commercial off-the-shelf (COTS) software — Multiple copies of software are sold at much lower prices to a large number of buyers

Terminology

- Shrink-wrapped software — Earlier term for COTS software
  - Because of the box containing the material
  - Nowadays, often downloaded over the World Wide Web
- Clickware — Used for COTS software
- Open-source software — Developed and maintained by a team of volunteers and may be downloaded and used free of charge
  - Becoming extremely popular
  - E.g., Linux operating system, Firefox Web browser, and Apache Web server
  - The source code is available to all
  - With most commercial products only executable version is sold
  - Can be of high quality: Any user can scrutinize the source code and report faults
  - Linus's Law: "Given enough eyeballs, all bugs are shallow"
  - "Release early, release often"

Terminology

- Software — Consists of not just the code, but all the documentation that is an intrinsic component of every project
  - E.g., specification document, legal documents, manuals
- System analysis — First two phases of traditional software development (requirements and analysis phases)
- System design — Third phase (design phase)
- Product — A nontrivial piece of software
  - The end result of a process is a product
- System — The combined hardware and software
- Methodology or Paradigm — A component of the software process as a whole
  - E.g., object-oriented paradigm
  - Methodology — Science of methods
  - Paradigm — A model or a pattern

Terminology

- Technique — A component of a portion of the software process
  - E.g., coding techniques, documentation techniques
- Mistake, fault, failure, error: Defined in IEEE Standard 610.12
  - A programmer makes a mistake
  - The consequence of that mistake is a fault in the code
  - Executing the software product results in a failure (observed incorrect behavior)
  - An error is the amount by which a result is incorrect
- Defect — Generic term that refers to a fault, failure, or error
  - Minimize the use
- Bug — Term for a fault
  - Minimize the use

Ethical Issues

- Most societies for professionals have a code of ethics to which all its members must adhere
  - They express similar sentiments
  - Vital for the future of the professions to rigorously adhere to such codes
- Software Engineering Code of Ethics and Professional Practice
- Developed by two major societies for computer professionals
  - Computer Society of IEEE (IEEE-CS)
  - Association for Computing Machinery (ACM) — World's largest educational and scientific computing society, delivering resources that advance computing as a science and a profession
- The standard for teaching and practicing software engineering
- It is lengthy, and a short version also produced

Ethical Issues

- Software Engineering Code of Ethics and Professional Practice (Version 5.2)
- Recommended by the IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices
- Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

(1) Public: Software engineers shall act consistently with the public interest.

(2) Client and Employer: Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.

Ethical Issues

(3) Product: Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

(4) Judgment: Software engineers shall maintain integrity and independence in their professional judgment.

(5) Management: Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

(6) Profession: Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

Ethical Issues

(7) Colleagues: Software engineers shall be fair to and supportive of their colleagues.

(8) Self: Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.