The Scope of Software Engineering
Xiaojun Qi

Software Engineering

- Software engineering is a discipline whose aim is the production of fault-free software, delivered on time and within budget, that satisfies the client’s need.

→ The software must be easy to modify when the user’s needs change.

Software

- Software consists of not just code in machine-readable form but also all the documentation that is an intrinsic component of every project.
  - The specification document
  - The design document
  - Legal and accounting documents of all kinds
  - The software project management plan and other management documents
  - All types of manuals.

People: Three Parties Involved

- **Client**: The client is the individual who wants a product to be built (developed).
- **Developers**: The developers are the members of a team responsible for building that product.
- **User**: The user is the person or persons on whose behalf the client has commissioned the product and who will utilize the software.

Two Types of Software: Based on the Parties’ Relationship

- **Internal Software**: Both the client and developers may be part of the same organization.

- **Contract Software**: The client and developers are members of totally independent organizations.

Three Types of Software: Based on the Functionality

- **Custom software**: It is written for one client.

- **Commercial off-the-shelf (COTS) software**: It has multiple copies and the copies are sold at much lower prices to a large number of buyers. It is developed for “the market”. That is, there are no specific clients or users until the software has been developed and is available for purchase.
  - Shrink-wrapped software
  - Clickware

- **Open-source software**: It is developed and maintained by a team of volunteers and may be downloaded and used free of charge by anyone.
The Six Phases in the Classical Software Engineering Life-Cycle Model

- A life-cycle model is a description of the steps that should be performed when building a software product.
- The overall life-cycle model is broken into a series of smaller steps, called phases.
- A life-cycle is the actual series of steps performed on a specific software product, from concept exploration through final retirement.
- There are six phases in the classical life-cycle model (i.e., waterfall model).

The Six Phases (Cont.)

- Requirements Phase: The concept is explored and refined, and the client's requirements are elicited.
- Analysis (Specification) Phase: The client's requirements are analyzed and presented in the form of the specification document. A software project management plan should be produced to describe the proposed software development in detail.
- Design Phase: Describe how the product does it!
  - Architecture design: The product as a whole is broken down into components, called modules.
  - Detailed design: Each module is then designed.

The Six Phases (Cont.)

- Implementation Phase:
  - The various components undergo coding and testing (unit testing) separately.
  - Integration: The components of the product are combined and tested as a whole.
  - Acceptance Testing: The client tests the product functions when the developers are satisfied with the product.
  - The implementation phase ends when the product is accepted by the client and installed on the client's computer.

The Six Phases (Cont.)

- Postdelivery Maintenance Phase: It includes all changes to the product once the product has been delivered and installed on the client's computer and passes its acceptance test.
  - Corrective Maintenance: Remove residual faults while leaving the specifications unchanged.
  - Enhancement: Change the specification and implement such changes.
    - Perfective maintenance: Improve the effectiveness of the product, such as additional functionality or decreased response time.
    - Adaptive maintenance: Respond to changes in the environment in which the product operates, such as a new hardware/operating system or new government regulations.

The Six Phases (Cont.)

- Retirement: It occurs when the product is removed from service. This occurs when the functionality provided by the product no longer is of any use to the client organization.

The Scope of Software Engineering

- The scope of software engineering is extremely broad. In general, five aspects are involved:
  - Historical Aspects
  - Economic Aspects
  - Maintenance Aspects
  - Requirements, Analysis, and Design Aspects
  - Team Development Aspects

- These five aspects can be categorized in the fields of Mathematics, Computer Science, Economics, Management, and Psychology.
The Scope of Software Engineering: Historical Aspects

- Software engineering cannot be considered as engineered since an unacceptable large proportion of software products still are being:
  - Delivered late
  - Over budget
  - With residual faults.
- Solution: A software engineer has to acquire a broad range of skills, both technical and managerial. These skills have to be applied to:
  - Programming; and
  - Every step of software production, from requirements to postdelivery maintenance.

The Scope of Software Engineering: Economic Aspects

- Applying economic principles to software engineering requires the client to choose techniques that reduce long-term costs in terms of the economic sense:
  - The cost of introducing new technology into an organization
    - Training cost
    - A steep learning curve
    - Unable to do productive work when attending the class.
  - The maintenance consequence

The Scope of Software Engineering: Maintenance Aspects

- Classical View of Maintenance:
  Development-then-maintenance model.
- However, this model is unrealistic due to:
  - During the development, the client’s requirements may change. This leads to changes in the specification and design.
  - Developers try to reuse parts of existing software products in the software product to be constructed.

The Scope of Software Engineering: Maintenance Aspects (Cont.)

- Modern view of Maintenance: It 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".
- That is, maintenance occurs whenever a fault is fixed or the requirements change, irrespective of whether this takes place before or after installation of the product.

The Scope of Software Engineering: Maintenance Definition

- Classical Postdelivery Maintenance: All changes to the product once the product has been delivered and installed on the client’s computer and passes its acceptance test.
- Modern Maintenance (or just maintenance): Corrective, perfective, or adaptive activities performed at any time.
- Classical postdelivery maintenance is a subset of modern maintenance.

The Scope of Software Engineering: Maintenance Aspects (Cont.)

- The importance of Postdelivery Maintenance:
  - A software product is a model of the real world, and the real world is perpetually changing. As a consequence, software has to be maintained constantly for it to remain an accurate reflection of the real world.
- A major aspect of software engineering consists of techniques, tools, and practices that lead to a reduction in postdelivery maintenance cost.
The Scope of Software Engineering: Requirements, Analysis, and Design Aspects

- The earlier we correct a fault, the better. That is, the cost of correcting a fault increases steeply since it is directly related to what has to be done to correct a fault.
  - If the mistake is made while eliciting the requirements, the resulting fault will probably also appear in the specifications, the design, and the code. Edit the code, recompile and relink the code, and test.
  - It is crucial to check that making the change has not created a new problem elsewhere in the product. All the relevant documentation, including manuals, needs to be updated.
  - The corrected product must be delivered and reinstalled.

Non-existence of Planning Phase

- There is no separate planning phase. Instead, planning activities are carried out all through the life cycle. Three types of planning activities may predominate at times:
  - Preliminary planning: It takes place for managing the requirements and analysis phases.
  - Software project management plan (SPMP): It includes the budget, staffing requirements, and detailed schedule.
  - All through the project, management needs to monitor the SPMP and be on the watch for any deviation from the plan.

Non-existence of Testing Phase

- Testing should be carried out constantly throughout every phase of the product development and maintenance process.
- There are times in the process when testing is carried out almost to the total exclusion of other activities. This occurs toward the end of each phase (verification) and is especially true before the product is handed over to the client (validation).
- Every software development organization should contain an independent group whose primary responsibility is to ensure that the delivered product is what the client needs and that the product has been built correctly in every way. This group is called the software quality assurance (SQA) group.

Non-existence of Documentation Phase

- The documentation of a software product must be complete, correct, and up to date. This is mainly because:
  - There is a large turnover in personnel in the software industry.
  - It is almost impossible to perform the steps of a specific phase unless the documentation of the previous phase is complete, correct, and up to date.
  - It is virtually impossible to test whether a software product is working correctly unless documents are available that state how that software product is supposed to behave.
  - Maintenance is almost impossible unless there is a complete and correct set of documentation that describes precisely what the current version of the product does.

The Classical Paradigm

- The classical (or structural) paradigm has been proved to be somewhat less successful in two respects:
  - The technique sometimes was unable to cope with the increasing size of software products.
  - The classical paradigm did not live up to earlier expectations during postdelivery maintenance.
- A major reason for the limited success of the classical paradigm is that classical techniques are either operation oriented or attribute (data) oriented but not both.
The Object-Oriented Paradigm

- The object-oriented paradigm considers both attributes and operations to be equally important.
- The advantages of using object-oriented paradigm:
  - Implementation details are local to an object
  - It makes maintenance quicker and easier, and the chance of introducing a regression fault is greatly reduced.
  - It makes development easier.
  - Well-designed objects are independent units. Everything in the product that relates to the portion of the real world modeled by that object can be found in the object itself.
  - Its resulting product consists of a number of smaller, largely independent units. It reduces the level of complexity of a software product and hence simplifies both development and maintenance.
  - It promotes reuse because objects are independent entities.

The Six Phases in Object-Oriented Paradigm

- Requirements Workflow
- Object-oriented analysis workflow
- Object-oriented design workflow
- Object-oriented implementation workflow
- Postdelivery maintenance
- Retirement

Differences Between the Classical and OO Paradigm

Classical:
Analysis (Specification) Phase:
- Determine what the product is to do

Object-oriented analysis workflow
- Determine what the product is to do
- Extract the classes (Architectural Design)

Differences Between the Classical and OO Paradigm (Cont.)

Classical:
Design Phase:
- Architecture design (Extract the modules)
- Detailed design

Object-oriented design workflow
- Detailed design

Differences Between the Classical and OO Paradigm (Cont.)

Classical:
Implementation Phase:
- Code the modules in an appropriate programming language
- Integrate

Object-oriented implementation workflow
- Code the classes in an appropriate object-oriented programming language
- Integrate

Ethical Issues

Most societies for professionals have a code of ethics to which all its members must adhere.

- Software engineers shall adhere to the following 8 principles:
  - Public
  - Client and employer
  - Product
  - Judgment
  - Management
  - Profession
  - Colleagues
  - Self