Unit-I

Software Engineering: An Introduction and Process Models

What is Software Engineering?

Software engineering is an engineering discipline concerned with all aspects of software production, from the early stages of system specification to maintaining the system after deployment. **It is crucial because the economies of all developed nations rely on software.** Software costs often exceed hardware costs, especially over a product's lifespan due to maintenance. **Therefore, software engineering focuses on cost-effective development.**

It's important to distinguish software engineering from computer science:

- **Computer science** focuses on the theory and fundamentals.
- **Software engineering** is concerned with the practicalities of creating and delivering successful software.

Similarly, software engineering is a part of the broader discipline of system engineering:

- **System engineering** encompasses all aspects of computer-based systems, including hardware, software, and process engineering.

Software Products and Their Attributes

There are two primary types of software products:

1. **Generic products:** Stand-alone systems sold to any customer, like PC software (graphics programs, project management tools) or specialized software for specific markets (appointment systems for dentists). In this case, the software developer owns the specifications and decides on changes.
2. **Customized products:** Software commissioned by a specific customer to meet their unique needs, such as embedded control systems or air traffic control software. Here, the customer owns the specification and makes decisions on necessary changes.

Regardless of the type, good software should exhibit these **essential attributes:**

- **Maintainability:** Written to easily evolve and meet changing needs in a dynamic business environment.
- **Dependability and security:** This includes reliability, safety, and security. The software shouldn't cause physical or economic damage, and it should protect against unauthorized access.
- **Efficiency:** Should not waste system resources like memory, processing time, etc., ensuring system responsiveness.
- **Acceptability:** Must be understandable, usable, and compatible with other systems for its target users.

Software Process Activities

Software development involves several key activities:

1. **Software specification:** Customers and engineers collaborate to define the software's functionality and operational constraints.
2. **Software development:** The software is designed and programmed.
3. **Software validation:** The software undergoes checks to ensure it meets the customer's requirements.
4. **Software evolution:** The software is modified over time to reflect changing customer and market demands.

Challenges in Software Engineering

Software development faces numerous challenges:

- **Heterogeneity:** Systems need to operate as distributed systems across networks with diverse computers and mobile devices.
- **Business and social change:** Rapid changes in business and society necessitate adapting existing software and quickly developing new software.
- **Security and trust:** Building trust and confidence in software is essential, given increasing security threats.
- **Software engineering diversity:** The vast array of software systems means no single set of techniques fits all. Methods and tools are chosen based on the application, customer requirements, and development team's expertise.

Types of Applications

Software engineering deals with diverse application types:

- **Stand-alone applications:** Run on a local computer (like a PC), containing all necessary functionality without network connectivity.
- **Interactive transaction-based applications:** Run on a remote computer and are accessed by users from their PCs or terminals, such as e-commerce web applications.
- **Embedded control systems:** Control and manage hardware devices.
- **Batch processing systems:** Process large amounts of data in batches, common in business systems.
- **Entertainment systems:** Primarily for personal use and entertainment.
- **Systems for modeling and simulation:** Used by scientists and engineers to model real-world processes or situations.
- **Data collection systems:** Collect data from the environment using sensors and transmit it to other systems for processing.
- **Systems of systems:** Composed of multiple other software systems.

Software Engineering and the Web

The Web has become a significant platform for running applications:

- Organizations increasingly develop web-based systems instead of local systems.
- Web services allow application functionality to be accessed remotely.
- Cloud computing, where applications run remotely on the cloud, has gained popularity, changing the model from buying software to paying per use.

Web-based software engineering presents unique considerations:

- **Software reuse** is the dominant approach, assembling systems from pre-existing components.
- **Incremental development and delivery** are crucial, as it's impractical to specify all requirements upfront.
- **User interfaces** are limited by web browser capabilities.

However, the fundamental principles of software engineering still apply to web-based systems.

Software Engineering Ethics

Software engineers have responsibilities extending beyond technical issues :

- **Confidentiality:** Respecting the confidentiality of their employers or clients, regardless of formal agreements.
- **Competence:** Not misrepresenting their skill level or accepting work exceeding their capabilities.
- **Intellectual property rights:** Being aware of and adhering to laws governing intellectual property (patents, copyright, etc.) to protect employers' and clients' rights.
- **Computer misuse:** Refraining from using technical skills to misuse others' computers, including seemingly trivial acts (game playing on work machines) and serious offences (virus dissemination).

Professional societies, like the ACM and IEEE, publish codes of conduct outlining expected behaviour for their members. The ACM/IEEE Code of Ethics emphasizes:

- The central role of computers and software engineers in diverse sectors.
- The potential for software engineers to do good or cause harm, directly or indirectly.
- The commitment to making software engineering beneficial and respected, prioritizing the health, safety, and welfare of the public.

The Code of Ethics outlines eight principles addressing various stakeholders:

1. **Public:** Acting consistently with the public interest.
2. **Client and Employer:** Acting in their best interests, aligned with the public interest.
3. **Product:** Ensuring products and modifications meet the highest professional standards.
4. **Judgment:** Maintaining integrity and independence in professional judgment.
5. **Management:** Promoting an ethical approach to software development management.
6. **Profession:** Advancing the integrity and reputation of the profession, consistent with the public interest.
7. **Colleagues:** Being fair and supportive.
8. **Self:** Participating in lifelong learning and promoting ethical approaches to the practice.

However, ethical dilemmas can arise, such as:

- Disagreements with senior management's policies.
- Employers acting unethically (releasing a safety-critical system without proper testing).
- Participation in developing controversial systems (military weapons or nuclear systems).

Software Process Models

A software process is a structured set of activities required to develop a software system. While various software processes exist, they all share common activities:

- **Specification:** Defining what the system should do.
- **Design and implementation:** Defining the system's organization and implementing it.
- **Validation:** Checking if the software meets customer needs.
- **Evolution:** Adapting the system to changing customer needs.

A software process model is an abstract representation of these activities.

Software process descriptions often include:

- **Products:** The outcomes of a process activity.
- **Roles:** Responsibilities of individuals involved.
- **Pre- and post-conditions:** Statements that hold true before and after a process activity is enacted or a product is created.

Two main approaches to software processes exist:

1. **Plan-driven processes:** All activities are planned in advance, and progress is measured against this plan.
2. **Agile processes:** Planning is incremental, allowing for easier adaptation to changing customer requirements.

In practice, most processes combine elements of both approaches. **There is no universally 'right' or 'wrong' process.**

Here are common software process models:

- **Waterfall model:** A plan-driven model with separate, distinct phases of specification and development.
- **Incremental development:** Specification, development, and validation are intertwined, adaptable to either plan-driven or agile approaches.
- **Reuse-oriented software engineering:** Assembling the system from existing components, also adaptable to plan-driven or agile methodologies.

Waterfall Model

The Waterfall model consists of sequential phases:

1. **Requirements analysis and definition:** Defining the system's requirements.
2. **System and software design:** Designing the system architecture and software components.
3. **Implementation and unit testing:** Developing and testing individual units of code.
4. **Integration and system testing:** Combining and testing all system components.
5. **Operation and maintenance:** Deploying the system and addressing bugs or enhancements.

Main Drawback: Difficulty accommodating change once the process is underway, as each phase needs completion before moving to the next.

Problems:

- **Inflexible partitioning** makes it difficult to respond to evolving customer requirements. Therefore, it's suitable only when requirements are well-understood and changes are minimal.
- Few real-world business systems have static requirements.

Usage: Mostly in large systems engineering projects developed at multiple sites.

Incremental Development

This model starts with an outline description and iteratively develops the software in increments (versions).

Benefits:

- **Reduced cost of accommodating change:** Less analysis and documentation need to be redone compared to the Waterfall model.
- **Easier customer feedback:** Customers can provide feedback on demonstrable software increments.
- **Faster delivery of usable software:** Increments provide value to customers sooner.

Problems:

- **Lack of process visibility:** Difficult for managers to measure progress without regular deliverables.
- **System structure degradation:** Regular changes without refactoring can make future changes costly and difficult.

Reuse-Oriented Software Engineering

This model relies on integrating existing components or COTS (Commercial-off-the-shelf) systems.

Process stages:

1. **Component analysis:** Analyzing available components.

2. **Requirements modification:** Adapting requirements to fit existing components.
3. **System design with reuse:** Designing the system around reusable components.
4. **Development and integration:** Developing the remaining components and integrating everything.

Reuse is standard practice in building many business systems.

Types of software components:

- **Web services:** Developed according to service standards, available for remote invocation.
- **Collections of objects:** Packaged for integration with frameworks like .NET or J2EE.
- **Stand-alone software systems (COTS):** Configured for a specific environment.

Coping with Change

Change is inevitable in large software projects due to:

- **Business changes:** Leading to new or modified system requirements.
- **New technologies:** Opening possibilities for improved implementations.
- **Changing platforms:** Requiring application modifications.

Change necessitates rework, incurring costs for both rework (e.g., re-analyzing requirements) and implementing new functionality.

Approaches to manage change:

- **Change avoidance:** Anticipating potential changes to minimize rework. For example, using prototypes to validate requirements with customers.
- **Change tolerance:** Designing the process to accommodate changes cost-effectively, usually through incremental development.

Software Prototyping

A prototype is an early version of a system used to showcase concepts and test design choices.

Uses:

- **Requirements engineering:** Eliciting and validating requirements.
- **Design:** Exploring options and developing user interface designs.
- **Testing:** Running back-to-back tests.

Process of prototype development:

1. **Establish prototype objectives:** Define the goals of the prototype.
2. **Define prototype functionality:** Determine the specific features to include.
3. **Develop prototype:** Build a working prototype, potentially with limited functionality.
4. **Evaluate prototype:** Test and gather feedback on the prototype.

Characteristics:

- Built using rapid prototyping languages or tools.
- May omit some functionality, focusing on poorly understood areas.
- May exclude error checking and recovery mechanisms.
- Prioritizes functional over non-functional requirements (reliability and security).

Throw-away prototypes: Discarded after development as they are unsuitable for production due to:

- Difficulties in tuning for non-functional requirements.
- Lack of documentation.
- Degraded structure from rapid changes.
- Not meeting organizational quality standards.

Incremental Delivery

Instead of a single delivery, development is split into increments, each delivering part of the functionality.

Key aspects:

- **Prioritized user requirements:** High-priority features are included in early increments.
- **Frozen requirements for each increment:** Once development begins, requirements are fixed, allowing later increments to incorporate evolving needs.

Incremental development and delivery are distinct but related:

- **Incremental development:** Developing the system in increments and evaluating each before moving to the next, common in agile methodologies. Evaluation is done by users or customer proxies.
- **Incremental delivery:** Deploying an increment for end-user use. This offers more realistic evaluation but can be challenging for replacement systems as increments have reduced functionality compared to the old system.

Advantages:

- **Faster value delivery:** Customers benefit from system functionality earlier.
- **Early prototypes:** Initial increments act as prototypes to refine requirements for later stages.
- **Lower risk of project failure:** Spreading the risk across increments.
- **Thorough testing for high-priority services:** As they are delivered in early increments.

Problems:

- **Identifying common facilities:** Challenging when requirements are incrementally defined, as it's difficult to ensure all increments can utilize shared functionalities.

- **Conflicts with traditional procurement:** Iterative development, where specifications evolve alongside the software, can clash with organizations requiring a complete system specification upfront in contracts.

Boehm's Spiral Model

This model visualizes the process as a spiral, not a linear sequence, acknowledging backtracking.

Key features:

- Each loop represents a phase.
- No fixed phases like specification or design; loops adapt to project needs.
- Explicit risk assessment and resolution throughout the process.

Spiral model sectors:

1. **Objective setting:** Defining specific objectives for each phase.
2. **Risk assessment and reduction:** Identifying and mitigating key risks.
3. **Development and validation:** Selecting a development model (any of the generic models) for the phase.
4. **Planning:** Reviewing and planning the next spiral loop.

Influence and Usage:

While highly influential in promoting iterative development and risk-driven approaches, the Spiral model is rarely used in its exact form for practical software development.

The Rational Unified Process (RUP)

The RUP is a modern, generic process model:

- Derived from work on the UML (Unified Modeling Language) and associated processes.
- Combines aspects of the waterfall, incremental, and iterative development models.

Three perspectives:

1. **Dynamic:** Showing phases over time.
2. **Static:** Illustrating process activities.
3. **Practical:** Suggesting best practices.

Phases:

1. **Inception:** Establishing the business case for the system.
2. **Elaboration:** Understanding the problem domain and defining the system architecture.
3. **Construction:** System design, programming, and testing.
4. **Transition:** Deploying the system into the operational environment.

Iterations:

- **In-phase iteration:** Each phase is iterative, with results delivered incrementally.
- **Cross-phase iteration:** The entire set of phases may be enacted incrementally.

Static workflows:

- **Business modelling:** Modeling business processes using use cases.
- **Requirements:** Identifying actors and developing use cases to model system requirements.
- **Analysis and design:** Creating a design model using architectural, component, object, and sequence models.
- **Implementation:** Structuring components into implementation sub-systems.
- **Testing:** An iterative process running alongside implementation, with system testing after implementation.
- **Deployment:** Creating, distributing, and installing a product release for users.
- **Configuration and change management:** Managing system changes.
- **Project management:** Overseeing system development.
- **Environment:** Providing appropriate software tools for the development team.

Good practices:

- Develop software iteratively, prioritizing customer needs.
- Explicitly document and track customer requirements and their changes.
- Utilize component-based architectures for reusability.
- Employ graphical UML models for static and dynamic software visualization.
- Ensure the software meets organizational quality standards.
- Manage software changes effectively using change management systems and configuration management tools.

Conclusion

Selecting the appropriate software process model depends on the project's specific needs and constraints. By understanding the characteristics, strengths, and weaknesses of each model, software engineers can make informed decisions to deliver high-quality software efficiently.

Question Bank

Software Engineering Basics

- What is software engineering and how does it relate to disciplines such as computer science and system engineering?
- Describe the essential attributes of good software.
- What are the high-level activities involved in all software processes?
- Explain the importance of software engineering in today's society, considering the economic implications and the increasing reliance on advanced software systems.
- Discuss the costs associated with software, particularly the costs of maintenance in comparison to development.

Software Products and Application Types

- Distinguish between generic and customised software products, providing examples of each type and explaining how product specification ownership differs between them.
- Describe various application types, including stand-alone applications, interactive transaction-based applications, embedded control systems, batch processing systems, entertainment systems, systems for modelling and simulation, data collection systems, and systems of systems.
- Explain how the Web has transformed software engineering, focusing on concepts like software reuse, incremental development and delivery, and the constraints of web browsers on user interfaces.

Software Processes

- What is a software process and what are its key elements?
- Compare and contrast plan-driven and agile processes, highlighting their differences in planning, adaptability to changing customer requirements, and real-world application.
- Describe the waterfall model of software development, outlining its phases, advantages, disadvantages, and applicability.
- Explain the concept of incremental development, including its benefits and problems, and discuss how it addresses the limitations of the waterfall model.
- Discuss reuse-oriented software engineering, detailing its process stages, the standard approach it represents for building business systems, and the different types of software components involved.
- How are the four basic process activities (specification, development, validation, and evolution) organised differently in various development processes, such as the waterfall model and incremental development?

Software Process Activities

- Outline the process of software specification, including the requirements engineering process with its steps like feasibility study, requirements elicitation and analysis, and requirements specification.
- Describe the software design and implementation process, differentiating between software design and implementation, and highlighting their interleaved nature.
- Explain the activities involved in design: architectural design, interface design, component design, and database design.

- What are the stages of software testing (component testing, system testing, and acceptance testing) and what is involved in each stage?
- How are testing phases integrated into a plan-driven software process?
- Discuss software evolution, explaining why it is an integral part of software development, and describe the steps involved in system evolution.

Coping with Change

- Why is change inevitable in large software projects and what are the sources and implications of such changes?
- Explain the approaches to reducing the costs of rework caused by change: change avoidance and change tolerance.
- What is software prototyping, how can it be used, and what are the steps involved in its development?
- Discuss the characteristics of prototype development, including the use of rapid prototyping tools and languages, the strategic omission of functionality, and the focus on specific areas and requirements.
- Why are throw-away prototypes generally discarded after development?
- Explain the concept of incremental delivery, distinguishing it from incremental development, and describe its advantages and problems.

Boehm's Spiral Model

- Describe Boehm's spiral model as an alternative representation of the software process, highlighting its key features: spiral representation, phase representation in loops, flexibility in determining loops, and risk assessment.
- What are the sectors within the spiral model (objective setting, risk assessment and reduction, development and validation, and planning) and what activities occur in each sector?
- Discuss the influence and practical usage of the spiral model in the field of software development.

The Rational Unified Process

- What is the Rational Unified Process (RUP), its origin, and the three generic process models it incorporates?
- Describe the phases (inception, elaboration, construction, and transition) in the RUP and the key activities associated with each phase.
- Explain the concepts of in-phase iteration and cross-phase iteration within the RUP.
- Outline the static workflows in the RUP: business modelling, requirements, analysis and design, implementation, testing, deployment, configuration and change management, project management, and environment.
- What are the good practices advocated by the RUP, covering aspects such as iterative development, requirements management, component-based architectures, visual modelling, software quality verification, and change control?

Ethical Considerations

- What are the key issues of professional responsibility for software engineers, particularly in regard to confidentiality, competence, intellectual property rights, and computer misuse?

- Describe the ACM/IEEE Code of Ethics, its rationale, the principles it contains, and the groups of software engineering professionals it addresses.
- What are some ethical dilemmas that software engineers might face in their professional practice?

Case Studies

- What are the three case studies presented in the source material, and what type of application does each represent?
- Analyse the case study of a personal insulin pump, considering its functionality, safety-critical aspects, hardware architecture, and activity model.
- Examine the case study of a mental health case patient management system, describing its purpose, features, concerns related to privacy and safety, and overall system organisation.
- Discuss the case study of a wilderness weather station, explaining its environment, its interaction with data management and station maintenance systems, and the additional software functionality required.

These questions encourage a deeper understanding of software engineering principles, processes, and ethical considerations. They require using information from the sources and combining it with analytical thinking and application to various scenarios.