Key Points

1 Introduction

- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Software is not just a program or programs but also includes documentation. Essential software product attributes are maintainability, dependability, security, efficiency, and acceptability.
- The software process includes all of the activities involved in software development. The high-level activities of specification, development, validation, and evolution are part of all software processes.
- The fundamental notions of software engineering are universally applicable to all types of system development. These fundamentals include software processes, dependability, security, requirements, and reuse.
- There are many different types of systems and each requires appropriate software engineering tools and techniques for their development. There are few, if any, specific design and implementation techniques that are applicable to all kinds of systems.
- The fundamental ideas of software engineering are applicable to all types of software systems. These fundamentals include managed software processes, software dependability and security, requirements engineering, and software reuse.
- Software engineers have responsibilities to the engineering profession and society. They should not simply be concerned with technical issues.
- Professional societies publish codes of conduct that set out the standards of behavior expected of their members.

2 Software Processes

- Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes. Examples of these general models include the waterfall model, incremental development, and reuse-oriented development.
- Requirements engineering is the process of developing a software specification. Specifications are intended to communicate the system needs of the customer to the system developers.
- Design and implementation processes are concerned with transforming a requirements specification into an executable software system. Systematic design methods may be used as part of this transformation.

- Software validation is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- Software evolution takes place when you change existing software systems to meet new requirements. Changes are continuous and the software must evolve to remain useful.
- Processes should include activities to cope with change. This may involve a prototyping phase that helps avoid poor decisions on requirements and design. Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.
- The Rational Unified Process is a modern generic process model that is organized into phases (inception, elaboration, construction, and transition) but separates activities (requirements, analysis, and design, etc.) from these phases.

3 Agile Software Development

- Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads, and producing high-quality code. They involve the customer directly in the development process.
- The decision on whether to use an agile or a plan-driven approach to development should depend on the type of software being developed, the capabilities of the development team, and the culture of the company developing the system.
- Extreme programming is a well-known agile method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement, and customer participation in the development team.
- A particular strength of extreme programming is the development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.
- The Scrum method is an agile method that provides a project management framework. It is centered around a set of sprints, which are fixed time periods when a system increment is developed. Planning is based on prioritizing a backlog of work and selecting the highest priority tasks for a sprint.
- Scaling agile methods for large systems is difficult. Large systems need up-front design and some documentation. Continuous integration is practically impossible when there are several separate development teams working on a project.

4 Requirements engineering

- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.

- Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used. These might be product requirements, organizational requirements, or external requirements. They often relate to the emergent properties of the system and therefore apply to the system as a whole.
- The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.
- The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification, requirements validation, and requirements management.
- Requirements elicitation and analysis is an iterative process that can be represented as a spiral of activities - requirements discovery, requirements classification and organization, requirements negotiation, and requirements documentation.
- Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism, and verifiability.
- Business, organizational, and technical changes inevitably lead to changes to the requirements for a software system. Requirements management is the process of managing and controlling these changes.

5 System modeling

- A model is an abstract view of a system that ignores some system details. Complementary system models can be developed to show the system's context, interactions, structure, and behavior.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes. They help define the boundaries of the system to be developed.
- Use case diagrams and sequence diagrams are used to describe the interactions between user and the system being designed and users/other systems. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a ssytem and their associations.
- Behavioral models are used to describe the dynamic behavior of an executing system. This can be modeled from the perspective of the data processed by the system or by the events that stimulate responses from a system.
- Activity diagrams may be used to model the processing of data, where each activity represents one process step.

- State diagrams are used to model a system's behavior in response to internal or external events.
- Model-driven engineering is an approach to software development in which a system is represented as a set of models that can be automatically transformed into executable code.

6 Architectural design

- A software architecture is a description of how a software system is organized. Properties of a system such as performance, security, and availability are influenced by the architecture used.
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used, and the ways in which the architecture should be documented and evaluated.
- Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used, and discuss its advantages and disadvantages.
- Commonly used architectural patterns include Model-View-Controller, Layered Architecture, Repository, Client-server, and Pipe and Filter.
- Generic models of application systems architectures help us understand the operation of applications, compare applications of the same type, validate application system designs, and assess large-scale components for reuse.
- Transaction processing systems are interactive systems that allow information in a database to be remotely accessed and modified by a number of users. Information systems and resource management systems are examples of transaction processing systems.
- Language processing systems are used to translate texts from one language into another and to carry out the instructions specified in the input language. They include a translator and an abstract machine that executes the generated language.

7 Design and Implementation

- Software design and implementation are interleaved activities. The level of detail in the design depends on the type of system being developed and whether you are using a plan-driven or agile approach.
- The process of object-oriented design includes activities to design the system architecture, identify objects in the system, describe the design using different object models, and document the component interfaces.

- A range of different models may be produced during an object-oriented design process. These include static models (class models, generalization models, association models) and dynamic models (sequence models, state machine models).
- Component interfaces must be defined precisely so that other objects can use them. A UML interface stereotype may be used to define interfaces.
- When developing software, you should always consider the possibility of reusing existing software, either as components, services, or complete systems.
- Configuration management is the process of managing changes to an evolving software system. It is essential when a team of people are cooperating to develop software.
- Most software development is host-target development. You use an IDE on a host machine to develop the software, which is transferred to a target-machine for execution.
- Open source development involves making the source code of a system publicly available. This means that many people can propose changes and improvements to the software.

8 Software Testing

- Testing can only show the presence of errors in a program. It cannot demonstrate that there are no remaining faults.
- Development testing is the responsibility of the software development team. A separate team should be responsible for testing a system before it is released to customers. In the user testing process, customers or system users provide test data and check that tests are successful.
- Development testing includes unit testing, in which you test individual objects and methods; component testing, in which you test related groups of objects; and system testing, in which you test partial or complete systems.
- When testing software, you should try to 'break' the software by using experience and guidelines to choose types of test cases that have been effective in discovering defects in other systems.
- Wherever possible, you should write automated tests. The tests are embedded in a program that can be run every time a change is made to a system.
- Test-first development is an approach to development where tests are written before the code to be tested. Small code changes are made and the code is re-factored until all tests execute successfully.
- Scenario testing is useful because it replicates the practical use of the system. It involves inventing a typical usage scenario and using this to derive test cases.
- Acceptance testing is a user testing process where the aim is to decide if the software is good enough to be deployed and used in its operational environment.

9 Software Evolution

- Software development and evolution can be thought of as an integrated, iterative process that can be represented using a spiral model.
- For custom systems, the costs of software maintenance usually exceed the software development costs.
- The process of software evolution is driven by requests for changes and includes change impact analysis, release planning, and change implementation.
- Lehman's laws, such as the notion that change is continuous, describe a number of insights derived from long-term studies of system evolution.
- There are three types of software maintenance, namely bug fixing, modifying the software to work in a new environment, and implementing new or changed requirements.
- Software re-engineering is concerned with restructuring and re-documenting software to make it easier to understand and change.
- Refactoring, making small program changes that preserve functionality, can be thought of as preventative maintenance.
- The business value of legacy system and the quality of the application software and its environment should be assessed to determine whether the system should be replaced, transformed, or maintained.

11 Dependability and Security

- Failure of critical computer systems can lead to large economic losses, serious information loss, physical damage, or threats to human life.
- The dependability of a computer system is a system property that reflects the user's degree of trust in the system. The most important dimensions of dependability are availability, reliability, safety, and security.
- The availability of a system is the probability that the system will be able to deliver services to its users when requested to do so. Reliability is the probability that the system services will be delivered as specified.
- Perceived reliability is related to the probability of an error occurring in operational use. A program may contain known faults but may still be experienced as reliable by its users. They may never use features of the system that are affected by the fault.
- The safety of a system is a system attribute that reflects the system's ability to operate, normally or abnormally, without injury to people or damage to the environment.
- Security reflects the ability of a system to protect itself against external attacks. Security failures may lead to loss of availability, damage to the system or its data, or the leakage of information to unauthorized people.

- Without a reasonable level of security, the availability, reliability, and safety of the system may be compromised if external attacks damage the system. If a system is unreliable, it is difficult to ensure system safety or security, as they may be compromised by system failures.

12 Dependability and Security Specification

- Risk analysis is an important activity in the specification of security and dependability requirements. It involves identifying risks that can result in accidents or incidents. System requirements are then generated to ensure that these risks do not occur and, if they do, that they do not lead to an incident or accident.
- A hazard-driven approach may be used to understand the safety requirements for a system. You identify potential hazards and decompose these (using methods such as fault tree analysis) to discover their root causes. You then specify requirements to avoid or recover from these problems.
- Reliability requirements can be defined quantitatively in the system requirements specification. Reliability metrics include probability of failure on demand (OPFOD), rate of occurrence of failure (ROCOF), and availability (AVAIL).
- It is important not to over-specify the required system reliability as this leads to unnecessary additional costs in the development and validation processes.
- Security requirements are more difficult to identify that safety requirements because a system attacker can use knowledge of system vulnerabilities to plan a system attack, and can learn about vulnerabilities from successful attacks.
- To specify security requirements, you should identify the assets that are to be protected and define how security techniques and technology should be used to protect these assets.
- Formal methods of software development rely on a system specification that is expressed as a mathematical model. Developing a formal specification has the key benefit of stimulating a detailed examination and analysis of the system requirements.

13 Dependability Engineering

- Dependability in a program can be achieved by avoiding the introduction of faults, by detecting and removing faults before system deployment, and by including fault-tolerance facilities that allow the system to remain operational after a fault has caused a system failure.
- The use of redundancy and diversity in hardware, software processes, and software systems is essential to the development of dependable systems.

- The use of a well-defined, repeatable process is essential if faults in a system are to be minimized. The process should include verification and validation activities at all stages, from requirements definition through to system implementation.
- Dependable system architectures are system architectures that are designed for fault tolerance. There are a number of architectural styles that support fault tolerance including protection systems, self-monitoring architectures, and N-version programming.
- Software diversity is difficult to achieve because it is practically impossible to ensure that each version of software is truly independent.
- Dependable programming relies on the inclusion of redundancy in a program to check the validity of inputs and the values of program variables.
- Some programming constructs and techniques, such as go-to statements, pointers, recursion, inheritance, and floating-point numbers, are inherently error prone. You should try to avoid these constructs when developing dependable systems.