## Chapter 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 all electronic documentation that is needed by system users, quality assurance staff, and developers. Essential software product attributes are maintainability, dependability and 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.
* There are many different types of system, and each requires appropriate software engineering tools and techniques for their development. Few, if any, specific design and implementation techniques are applicable to all kinds of system.
* The fundamental ideas of software engineering are applicable to all types of software system.
* 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 but should be aware of the ethical issues that affect their work.
* Professional societies publish codes of conduct that embed ethical and professional standards.
* These set out the standards of behavior expected of their members.

## Chapter 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 reusable component configuration and integration.
* 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.
* 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.
* Process improvement is the process of improving existing software processes to improve software quality, lower development costs, or reduce development time. It is a cyclic process involving process measurement, analysis, and change.

## Chapter 3: Agile software development

* Agile methods are iterative development methods that focus on reducing process overheads and documentation and on incremental software delivery. They involve customer representatives 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. In practice, a mix of agile and plan-based techniques may be used.
* Agile development practices include requirements expressed as user stories, pair programming, refactoring, continuous integration, and test-first development.
* Scrum is an agile method that provides a framework for organizing agile projects. 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.
* To scale agile methods, some plan-based practices have to be integrated with agile practice.
* These include up-front requirements, multiple customer representatives, more documentation, common tooling across project teams, and the alignment of releases across teams.

## Chapter 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 requirements engineering process includes requirements elicitation, requirements specification, requirements validation, and requirements management.
* Requirements elicitation 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 specification is the process of formally documenting the user and system requirements and creating a software requirements document.
* 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.
* 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.

## Chapter 5: System modelling

* A model is an abstract view of a system that deliberately 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 users and systems in the system being designed. 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 system and their associations.
* Behavioral models are used to describe the dynamic behavior of an executing system. This behavior 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 to executable code.

## Chapter 6: Architecture 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.
* Architectures may be documented from several different perspectives or views. Possible views include a conceptual view, a logical view, a process view, a development view, and a physical view.
* Architectural patterns are a means of reusing knowledge about generic system architectures.
* They describe the architecture, explain when it may be used, and point out 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.

## Chapter 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 is 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.

## Chapter 8: Software testing

* Testing can only show the presence of errors in a program. It cannot show 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 whereby tests are written before the code to be tested. Small code changes are made, and the code is refactored 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 in which the aim is to decide if the software is good enough to be deployed and used in its planned operational environment.

## Chapter 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.
* Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business.
* It is often cheaper and less risky to maintain a legacy system than to develop a replacement system using modern technology.
* The business value of a legacy system and the quality of the application software and its environment should be assessed to determine whether a system should be replaced, transformed, or maintained.
* There are three types of software maintenance, namely, bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.
* Software reengineering is concerned with restructuring and redocumenting software to make it easier to understand and change.
* Refactoring, making small program changes that preserve functionality, can be thought of as preventative maintenance.

–

## Chapter 10: Dependable systems

* System dependability is important because 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, security, and resilience.
* Sociotechnical systems include computer hardware, software, and people, and are situated within an organization. They are designed to support organizational or business goals and objectives.
* The use of a dependable, 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.
* The use of redundancy and diversity in hardware, software processes, and software systems is essential to the development of dependable systems.
* Formal methods, where a formal model of a system is used as a basis for development, help reduce the number of specification and implementation errors in a system. However, formal methods have had a limited take-up in industry because of concerns about the cost-effectiveness of this approach.

## Chapter 11: Reliability engineering

* Software reliability 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.
* Reliability requirements can be defined quantitatively in the system requirements specification.
* Reliability metrics include probability of failure on demand (POFOD), rate of occurrence of failure (ROCOF), and availability (AVAIL).
* Functional reliability requirements are requirements for system functionality, such as checking and redundancy requirements, which help the system meet its non-functional reliability requirements.
* Dependable system architectures are system architectures that are designed for fault tolerance.
* A number of architectural styles support fault tolerance, including protection systems, selfmonitoring architectures, and N-version programming.
* Software diversity is difficult to achieve because it is practically impossible to ensure that each version of the software is truly independent.
* Dependable programming relies on including redundancy in a program as checks on the validity of inputs and the values of program variables.
* Statistical testing is used to estimate software reliability. It relies on testing the system with test data that matches an operational profile, which reflects the distribution of inputs to the software when it is in use.

## Chapter 12: Safety engineering

* Safety-critical systems are systems whose failure can lead to human injury or death.
* A hazard-driven approach may be used to understand the safety requirements for safety-critical systems.
* You identify potential hazards and decompose them (using methods such as fault tree analysis) to discover their root causes. You then specify requirements to avoid or recover from these problems.
* It is important to have a well-defined, certified process for safety-critical systems development.
* The process should include the identification and monitoring of potential hazards.
* Static analysis is an approach to V & V that examines the source code (or other representation) of a system, looking for errors and anomalies. It allows all parts of a program to be checked, not just those parts that are exercised by system tests.
* Model checking is a formal approach to static analysis that exhaustively checks all states in a ­system for potential errors.
* Safety and dependability cases collect all of the evidence that demonstrates a system is safe and dependable. Safety cases are required when an external regulator must certify the system before it is used.

## Chapter 15: Software reuse

* There are many different ways to reuse software. These range from the reuse of classes and methods in libraries to the reuse of complete application systems.
* The advantages of software reuse are lower costs, faster software development, and lower risks.
* System dependability is increased. Specialists can be used more effectively by concentrating their expertise on the design of reusable components.
* Application frameworks are collections of concrete and abstract objects that are designed for reuse through specialization and the addition of new objects. They usually incorporate good design practice through design patterns.
* Software product lines are related applications that are developed from one or more base applications. A generic system is adapted and specialized to meet specific requirements for functionality, target platform, or operational configuration.
* Application system reuse is concerned with the reuse of large-scale, off-the-shelf systems.
* These provide a lot of functionality, and their reuse can radically reduce costs and development time. Systems may be developed by configuring a single, generic application system or by integrating two or more application systems.
* Potential problems with application system reuse include lack of control over functionality, performance, and system evolution; the need for support from external vendors; and difficulties in ensuring that systems can interoperate.

## Chapter 18: Service-oriented software engineering

* Service-oriented architecture is an approach to software engineering where reusable, standardized services are the basic building blocks for application systems.
* Services may be implemented within a service-oriented architecture using a set of XML-based web service standards. These include standards for service communication, interface definition, and service enactment in workflows.
* Alternatively, a RESTful architecture may be used, which is based on resources and standard operations on these resources. A RESTful approach uses the http and https protocols for service communication and maps operations on the standard http verbs POST, GET, PUT, and DELETE.
* Services may be classified as utility services that provide a general-purpose functionality, business services that implement part of a business process, or coordination services that coordinate the execution of other services.
* The service engineering process involves identifying candidate services for implementation, defining the service interface, and implementing, testing, and deploying the service.
* The development of software using services is based on the idea that programs are created by composing and configuring services to create new composite services and systems.
* Graphical workflow languages, such as BPMN, may be used to describe a business process and the services used in that process. These languages can describe interactions between the organizations that are involved.

## Chapter 19: Systems engineering

* Systems engineering is concerned with all aspects of specifying, buying, designing, and testing complex sociotechnical systems.
* Socio-technical systems include computer hardware, software, and people, and are situated within an organization. They are designed to support organizational or business goals and objectives.
* The emergent properties of a system are characteristics of the system as a whole rather than of its component parts. They include properties such as performance, reliability, usability, safety, and security.
* The fundamental systems engineering processes are conceptual systems design, system procurement, system development, and system operation.
* Conceptual systems design is a key activity where high-level system requirements and a vision of the operational system is developed.
* System procurement covers all of the activities involved in deciding what system to buy and who should supply that system. Different procurement processes are used for off-the-shelf application systems, configurable COTS systems, and custom systems.
* System development processes include requirements specification, design, construction, integration, and testing.
* When a system is put into use, the operational processes and the system itself inevitably change to reflect changes to the business requirements and the system’s environment.