What is software?

Answer: Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.

What is software engineering?

Answer: Software engineering is an engineering discipline which is concerned with all aspects of software production.

What is the difference between software engineering and computer science?

Answer: Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

What is the difference between software engineering and system engineering?

Answer: System engineering is concerned with all aspects of computer-based systems development, including hardware,- software and process engineering. Software engineering is part of this process.

What is a software process?

Answer: A set of activities whose goal is the development or evolution of software.

What is a software process model?

Answer: A simplified representation of a software process, presented from a specific perspective.

What are the costs of software engineering?

Answer: Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.

What are software engineering methods?

Answer: Structured approaches to software development which include system models, notations, rules, design advice and process guidance.

What is CASE (Computer-Aided Software Engineering)?

Answer: Software systems which are intended to provide automated support for software process activities. CASE systems are often used for method support.

What are the attributes of good software?

Answer: The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

What are the key challenges facing software engineering?

Answer: Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.

1.1.2 What is software engineering?

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.

1. *Engineering discipline* Engineers make things work. They apply theories, methods and tools where these are appropriate, but they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognise that they must work to organisational and financial constraints, so they look for solutions within these constraints.
2. *All aspects of software production* Software engineering is not just concerned with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production.

1.1.3 What's the difference between software engineering and computer science?

Essentially, computer science is concerned with the theories and methods that underlie computers and software systems, whereas software engineering is concerned with the practical problems of producing software. Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers.

1.1.4 What is the difference between software engineering and system engineering?

System engineering is concerned with all aspects of the development and evolution of complex systems where software plays a major role. System engineering is there-fore concerned with hardware development, policy and process design and system deployment as well as software engineering. System engine rs arc involved in specifying the system, defining its overall architecture and integrating the different parts to create the finished system. They are less con ed with the engineering of the system components (hardware, software, etc.).

1.1.5 What is a software process?

A software process is the set of activities and associated results that produce a soft-ware product. There are four fundamental process activities (covered later in the book) that are common to all software processes. These are:

1. Software specification where customers and engineers define the software to be produced and the constraints on its operation.
2. Software development where the software is designed and programmed.
3. Software validation where the software is checked to ensure that it is what the customer requires.
4. Software evolution where the software is modified to adapt it to changing customer and market requirements.

Different types of systems need different development processes. For example, real-time software in an aircraft has to be completely specified before development begins whereas, in e-commerce systems, the specification and the program are usu-ally developed together. Consequently, these generic activities may be organised in different ways and described at different levels of detail for different types of soft-ware. However, use of an inappropriate software process may reduce the quality or the usefulness of the software product to be developed and/or increase the development costs.

1.1.6 "What is a software process model?

A software process model is a simplified description of a software process that presents one view of that process. Process models may include activities that are part of the software process, software products and the roles of people involved in soft-ware engineering. Some examples of the types of software process model that may be produced are:

1. A *workflow* model This shows the sequence of activities in the process along with their inputs, outputs and dependencies. The activities in this model rep-resent human actions.
2. A *dataflow* or activity model This represents the process as a set of activities. each of which carries out some data transformation. It shows how the input to the process, such as a specification, is transformed to an output. such as a design. The activities here may represent transformations carried cut by people or by computers.
3. A *role/action* to This represents the roles of the people involved in the soft-ware process and the activities for which they are responsible.

Most software process models are based on one of three general models or paradigms of software development:

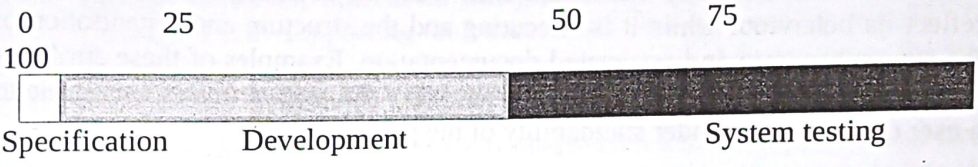
1. The *waterfall approach* This takes the above activities and represents them as separate process phases such as requirements specification. software design. implementation. testing and so on. After each stage is defined it is 'signed-off , and development goes on to the following stage.
2. *Iterative development* This approach interleaves the activities of specification. development and validation. An initial system is rapidly developed from very abstract specifications. This is then refined with customer input to produce a system that satisfies the customer's needs. The system may then be delivered. Alternatively, it may be reimplemented using a more structured approach to produce a more robust and maintainable system.
3. Component-based software engineering (CBSE) This technique assumes that parts of the system already exist.

1.1.7 What are the costs of software engineering?

There is no simple answer to this question as the distribution of costs across the different activities in the software process depends on the process used and the type of software that is being developed. For example, real-time software usually requires more extensive validation and testing than web-based systems. However, each of the different generic approaches to software development has a different profile of cost distribution across the software process activities.

1.1.8 What are software engineering methods?

A software engineering method is a structured approach to software development whose aim is to facilitate the production of high-quality software in a cost-effective way. Methods such as Structured Analysis (DeMarco, 1978) and JSD (Jackson, 1983) were first developed in the 1970s. These methods attempted to identify the basic functional components of a system; function-oriented methods are still used. In the 1980s and 1990s, these function-oriented methods were supplemented by object-oriented (00) methods such as those proposed by Booch (Booch, 1994) and Rumbaugh (Rumbaugh, et al., 1991). These different approaches have now been integrated into a single unified approach built around the Unified Modeling Language (UML) (Booch, et al., 1999; Rumbaugh, et al., 1999a; Rumbaugh, et al., 1999b).



Component	Description	Example
System model descriptions	Descriptions of the system models which should be developed and the notation used to define these models.	Object models, data-flow models, state machine models, etc.
Rules	Constraints which always apply to system models.	Every entity in a system model must have a unique name.
Recommendations	Heuristics which characterise good design practice in this method. Following these recommendations should lead to a well-organised system model.	No object should have more than seven sub-objects associated with it.
Process guidance	Descriptions of the activities which may be followed to develop the system models and the organisation of these activities.	Object attributes should be documented before defining the operations associated with an object.

There is no ideal method, and different methods have different areas where they are applicable. For example, object-oriented methods are often appropriate for interactive systems but not for systems with stringent real-time requirements.

1.1.9 What is CASE?

The acronym CASE stands for Computer-Aided Software Engineering. It covers a wide range of different types of programs that are used to support software process activities such as requirements analysis, system modelling, debugging and testing. All methods now come with associated CASE technology such as editors for the notations used in the method, analysis modules which check the system model according to the method rules and report generators to help create system documentation.

1.1.10 What are the attributes of good software?

As well as the services that it provides, software products have a number of other associated attributes that reflect the quality of that software. These attributes are not directly concerned with what the software does. Rather, they reflect its behaviour while it is executing and the structure and organisation of the source program and associated documentation. Examples of these attributes (sometimes called non-functional attributes) are the software's response time to a user query and the under standability of the program code.

The specific set of attributes that you might expect from a software system obviously depends on its application. Therefore, a banking system must be secure, an

Product characteristics	Description
Maintainability	Software should be written in such a way that it may evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable consequence of a changing business environment.
Dependability	Software dependability has a range of characteristics. including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Usability	Software must be usable, without undue effort, by the type of user for whom it is designed. This means that it should have an appropriate user interface and adequate documentation.

interactive game must be responsive,a telephone switching system must be reliable and so on. These can be generalised into the set of attributes shown in Figure 1.5,Which, I believe, are the essential characteristics of a well-designed software system.

1.1.11 What are the key challenges facing software engineering?

Software engineering in the 21st century faces three key challenges:

1. *The heterogeneity challenge* Increasingly, systems are required to operate as distributed systems across networks that include different types of computers and with different kinds of support systems. It is often necessary to integrate new software with older legacy systems written in different programming languages. The heterogeneity challenge is the challenge of developing techniques for building dependable software that is flexible enough to cope with this heterogeneity.
2. *The delivery challenge* Many traditional software engineering techniques are time-consuming. The time they take is required to achieve software quality. However, businesses today must be responsive and change very rapidly. Their supporting software must change equally rapidly. The delivery challenge is the challenge of shortening delivery times for large and complex systems without compromising system quality.
3. *The trust challenge* As software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface. The trust challenge is to develop techniques that demonstrate that software can be trusted by its users.

A Generic View of Software Engineering

Engineering is the analysis, design, construction, verification, and management of technical (or social) entities. Regardless of the entity to be engineered, the following questions must be asked and answered:

- What is the problem to be solved?
- What characteristics of the entity are used to solve the problem?
- How will the entity (and the solution) be realized?
- How will the entity be constructed?
- What approach will be used to uncover errors that were made in the design and construction of the entity?
- How will the entity be supported over the long term,when corrections, adaptations, and enhancements are requested by users of the entity.

The work associated with software engineering can be categorized into three generic phases,regardless of application area, project size, or complexity. Each phase addresses one or more of the questions noted previously.

The *definition phase* focuses on what. That is,during definition,the software engineer attempts to identify what information is to be processed, what function and performance are desired,what system behavior can be expected, what interfaces are to be established,what design constraints exist, and what validation criteria are required to define a successful system. The key requirements of the system and the software are identified. Although the methods applied during the definition phase will vary depending on the software engineering paradigm (or combination of paradigms) that is applied, three major tasks will occur in some form: system or information engineering, software project planning, and requirements analysis.

The *development phase* focuses on how. That is, during development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how procedural details are to be implemented, how interfaces are to be characterized,how the design will be translated into a programming language (or nonprocedural language), and how testing will be performed. The methods applied during the development phase will vary, but three specific technical tasks should always occur:software design, code generation,and software testing.

The *support phase* focuses on change associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. The support phase reapplies the steps of the definition and development phases but does so in the context of existing software. Four types of change are encountered during the support phase:

Correction. Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. Corrective maintenance changes the software to correct defects.
Adaptation. Over time,the original environment (e.g.,CPU,operating system, the software was developed is likely to change. Adaptive maintenance results in modification to the software to accommodate changes to its external environment.

Enhancement. As software is used, the customer/user will recognize additional functions that will provide benefit. *Perfective* maintenance extends the software beyond its original functional requirements.

Prevention. Computer software deteriorates due to change,and because of this, *preventive maintenance*, often called *software reengineering*, must be conducted to enable the software to serve the needs of its end users. In essence, preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted, and enhanced.

In addition to these support activities, the users of software require continuing sup-port. In-house technical assistants, telephone-help desks, and application-specific Web sites are often implemented as part of the support phase.

The phases and related steps described in our generic view of software engineering are complemented by a number of umbrellas activities. Typical activities in this category include:

- Software project tracking and control
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Document preparation and production
- Reusability management
- Measurement
- Risk management

2.2 THE SOFTWARE PROCESS

A software process can be characterized as shown in Figure 2.2. A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. A number of task sets-each a collection of software engineering work tasks, project milestones, work products, and quality assurance points-enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. Finally, umbrella activities-such as software quality assurance, soft-ware configuration management, and measurement2-overlay the process model. Umbrella activities are independent of any one framework activity and occur through-out the process.

The SEI approach provides a measure of the global effectiveness of a company's software engineering practices and establishes five process maturity levels that are defined in the following manner:

Level 1: Initial. The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

Level 2:Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is ,in place to repeat earlier successes on projects with similar applications.

Level 3:Defined. The software process for both management and engineering activities is documented,standardized,and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

Level 4:Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level 5:Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

The five levels defined by the SEI were derived as a consequence of evaluating responses to the SEI assessment questionnaire that is based on the CMM. The results of the questionnaire are distilled to a single numerical grade that provides an indication of an organization's process maturity.

- The SEl has associated key process areas(KPAs) wih each of the maturity levels. The KPAs describe those software engineering functions (e.g., software project planning, requirements management) that must be present to satisfy good practice at a particular level. Each KPA is described by identifying the following characteristics:
- Goals--the overall objectives that the KPA must achieve.
- Commitments-requirements (imposed on the organization) that must be met to achieve the goals or provide proof of intent to comply with the goals.
- Abilities--those things that must be in place (organizationally and technically) to enable the organization to meet the commitments.
- Activities-the specific tasks required to achieve the KPA function.
- Methods for monitoring implementation-the manner in which the activities are monitored as they are put into place.
- Methods for verifying implementation-the manner in which proper practice for the KPA can be verified.

Process maturity level 2

- Software configuration management
- Software subcontract management
- Software project tracking and oversight
- Software project planning
- Requirements management

Process maturity level 3

- Peer reviews
- Intergroup coordination
- Software product engineering
- integrated software management
- Training program
- Organization process definition
- Organization process focus

Process maturity level 4

- Software quality management
- Quantitative process management

Process maturity level 5

- Process change management
- Technology change management
- Defect prevention