1.2.2 Software Characteristics

Different individuals judge software on different basis. This is because they are involved with the software in different ways. For example, users want the software to perform according to their requirements. Similarly, developers involved in designing, coding and maintenance of the software evaluate the software by looking at the internal characteristics of the products, before delivering it to the user. Software characteristics are classified into six major components. These components are listed below:

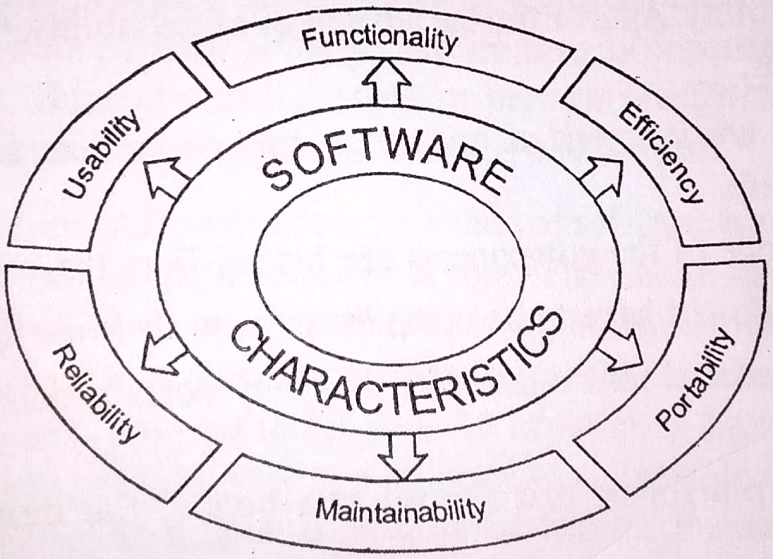


Figure 1.1 Characteristics of Software

- **Functionality:** Refers to the degree of performance of the software against its intend purpose.
- **Reliability:** Refers to the ability of software to perform a required function unde given conditions for a specified period.
- **Usability:** Refers the degree to which software is easy to use.
- **Efficiency:** Refers to the ability of software to use system resource in the most effectiv and efficient manner.
- **Maintainability:** Refers to the ease with which a software system can be modified t add capabilities, improve system performance, or correct errors.
- **Portability:** Refers to the ease with which software developers can transfer softwar from one platform to another, without (or with minimum) changes. In simple terms, refers to the ability of software to function properly on different hardware and softwar platforms without making any changes in it.

In addition to the above-mentioned characteristics, *robustness* and *integrity* are also considered to be important. Robustness refers to the extent to which software can continue to operate correctly despite the introduction of invalid input, while integrity refers to the extent to which unauthorised access or modification of software or data can be controlled in the computer system.

1.2.3 Software Components

A software component is defined as an independent executable software element with a well-defined set of inputs, outputs and interface. All the services provided by a componen are made available through the interface and all the interactions with the component are done through that interface. Council and Heinmann define the term software component as follows.

"A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition."

Szyperski describes the term as follows.

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

The software engineering that emphasizes the design and development of computer-based systems using software components is called component-based software engineering (CBSE). The main objective of CBSE is to standardize the interfaces between software components so that they can be assembled easily to develop new software. The basic idea behind CBSE is to reuse the existing components. The components developed for a specific application have to be generalized to make them reusable. In other words, the more generalized interface, the greater the reusability. Apart from the advantage of reusability, the components have the following advantages.

- The components are independent and hence, they do not interfere with each other and are easy to modify.
- The inner workings of the components are hidden from the user.
- The components do not have to be compiled prior to their use with other components.
- The communication and interaction with the components is done through well-defined interfaces.
- The component platforms are shared and hence, the development costs are reduced.

There are some characteristics that a software program must possess before it qualifies as a component. These are given below.

- It should be independent and deployable, that is, it has to be a self-contained and stand-alone entity and it should not depend on other software components for its use.
- It should provide some pre-defined interfaces and all interactions must take through these interfaces.
- It should have a complete documentation so that the users of the component can decide whether or not the component is meeting their needs.
- It has to conform to some specified standards.
- It should be language-independent.

Exam

1.2.3.1 Components Applications

Software can be applied in countless situations, such as in business, education, social sector, and in other fields. The only thing that is required is a defined set of procedural steps. That is, software can be engaged in any field, which can be described in logical and related steps. Every software is designed to suit some specific goals. These goals are data processing, information sharing, communication, and so on. Software is classified according to the range of potential applications. These classifications are listed below:

- **System software:** This class of software is responsible for managing and controlling operations of a computer system. System software is a group of programs rather than one program and is responsible for using computer resources efficiently and effectively. For example, operating system is system software, which controls the hardware, manages memory and multi-tasking functions, and acts as an interface between applications programs and the computer.

- **Real-time software:** This class of software observes, analyses, and controls real world events as they occur. Generally, a real-time system guarantees a response to an external event within a specified period of time. For example, real-time software is used for navigation in which the computer must react to a steady flow of new information without interruption. Most of the defence organisations all over the world use real-time software to control their military hardware.

- **Business software:** This class of software is widely used in areas where the management and control of financial activities is of utmost importance. The fundamental component of a business software comprises of payroll, inventory, accounting, and software that permits user to access relevant data from the database. These activities are usually performed with the help of specialised business software that facilitates efficient framework in the business operation and in management decisions.

- **Engineering and scientific software:** This class of software has emerged as a powerful tool to provide help in the research and development of next generation technology. Applications, such as study of celestial bodies, study of under-surface activities, and programming of orbital path for space shuttle are heavily dependent on engineering and scientific software. This software is designed to perform precise calculations on complex numerical data that are obtained during real-time environment.

- **Artificial intelligence (AI) software:** This class of software is used where the problem solving technique is non-algorithmic in nature. The solutions of such problems are generally non-agreeable to computation or straightforward analysis. Instead, these problems require specific problem solving strategies that include expert system, pattern recognition, and game playing techniques. In addition, it involves different kinds of searching techniques including the use of heuristics. The role of artificial intelligence software is to add certain degree of intelligence into the mechanical hardware to have the desired work done in an agile manner.

- **Web-based software:** This class of software acts as an interface between the user and the Internet. Data on the Internet can be in the form of text, audio, or video format, linked with hyperlinks. Web browser is a web-based software that retrieves web pages from the Internet. The software incorporates executable instructions written in special scripting languages, such as CGI or ASP. Apart from providing navigation on the web, this software also supports additional features that are useful while surfing the Internet.

- **Personal computer (PC) software:** This class of software is used for official and personal use on daily basis. The personal computer software market has grown over the last two decades from normal text editor to word processor and from simple paintbrush to advance image-editing software. This software is used predominantly in almost every field, whether it is database management system, financial accounting package, or a multimedia based software. It has emerged as a versatile tool for daily life applications.

Software can be also classified in terms of how closely software users or software purchasers are associated with the software development.

- **Commercial off the shelf (COTS):** In this category comes the software for which there is no committed user before it is put up for sale. The software users have less or no contact with the vendor during development. It is sold through retail stores or distributed electronically. This software includes commonly used programs, such as word processors, spreadsheets, games, income tax programs, as well as software development tools, such as, software testing tools and object modelling tools.
- **Customised or bespoke:** In this classification, software is developed for a specific user, who is bound by some kind of formal contract. For example, software developed for an aircraft is usually done for a particular aircraft making company. They are not purchased 'off-the-shelf' like any word processing software.
- **Customised COTS:** In this classification, user can enter into a contract with the software vendor to develop a COTS product for a special purpose, that is, software can be customised according to the needs of the user. Another growing trend is the development of COTS software components—components that are purchased and used to develop new applications.

The COTS software component vendors are essentially parts stores, which are classified according to their application types. These types are listed below:

- **Stand-alone Software:** Resides on a single computer and does not interact with any other software installed in a different computer.
- **Embedded Software:** Part of unique application involving hardware like automobile controller.
- **Real-time Software:** Operations execute within very short time limits, often microseconds. For example, radar software in air traffic control system.
- **Network Software:** Software and its components interact across a network.

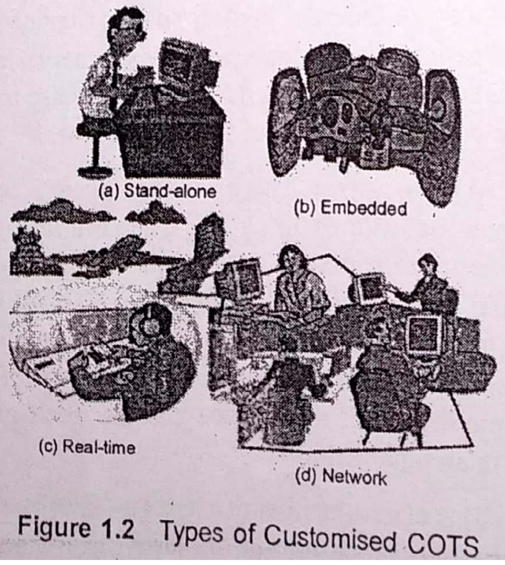


Figure 1.2 Types of Customised COTS

1.2.5 Evolution of Software Engineering

In the late 1960s, it was clear that software development was unlike the construction of physical structures. This was because in software development, more programmers could not be added simply to speed up a lagging development project. Software had become a critical component of many systems, yet it was too complex to develop with any certainty of schedule or quality. This problem imposed financial and public safety concerns.

Software errors have caused large-scale financial losses as well as inconvenience to many. Disasters, such as Y2K problem have affected economic, political, and administrative system of various countries around the world. This situation where catastrophic failures have occurred is known as **Software crisis**.

Software crisis is a term that has been used since the early days of software engineering to describe the impact of the rapid increases in computer power and its complexity. Software crisis occurs due to problems associated with poor quality software. This includes problems arising from malfunctioning of software systems, inefficient development of software, and most importantly, dissatisfaction among users of the software. Other problems associated with software are listed below:

- Software complexity can be managed by dividing the system into subsystems, but, as systems grow, the interaction between subsystems increases non-linearly. This leads to a situation where problem domain cannot be understood properly.
- It is difficult to establish an adequate and stable set of requirements for a software system. This is because hidden assumptions exist. In addition, there is no analytic procedure available for determining whether the developers are aware of the user's requirements or not, thus creating an environment where both users and developers are unaware of the requirements.

Software market today has a turnover of more than millions of rupees. Out of this, approximately 30% of software is used for personal computers and the remaining software is developed for specific users or organizations. Application areas, such as the banking sector are completely dependent on software application for their working. Software failures in these technology-oriented areas have led to considerable loss in terms of time, money, and even human lives. History has seen many such failures. Some of these are listed below:

- During the gulf war in 1991, United States of America used Patriot missile as a defence against Iraqi Scud missile. However, this Patriot failed to hit Scud missile many times. As a result 28 US soldiers were killed in Dhahran, Saudi Arabia. An inquiry into the incident concluded that a small bug resulted in the miscalculation of missile path.
- Arian-5 space rocket developed at the cost of $7000 million over a period of 10 years was destroyed in 39 seconds, after its launch. The crash occurred because a software bug existed in the rocket guidance system.
- In June 1980, the North American Aerospace Defence Command (NORAD) reported that the US was under missile attack. The report was traced to a faulty computer circuit that generated incorrect signals. If the developers of the software responsible for processing these signals had taken into account the possibility that the circuit could fail, the false alert might not have occurred.
- Year 2000 (Y2K) problem refers to the widespread snags computers had in processing dates after the year 2000. Seeds of the Y2K trouble were planted during 1960–80, when commercial computing was new and storing memory was relatively limited. The developers at that time shortened the 4-digit date format like 1994 to a 2-digit format, like 94. In the 1990s, experts began to realise this major shortcoming in the computer application and millions were spent to handle this problem.

Scan

1.3 SOFTWARE ENGINEERING: DEFINITION

As discussed earlier, over the last 50 years there has been a dramatic advancement in the field of technology, leading to improvements in hardware performance and profound changes in computing architectures. This advancement has led to the production of complex computer-based systems that are capable of providing information in a wide variety of formats. The increase in computer power has made unrealistic computer applications a feasible proposition, marking the genesis of an era where software products are far more complex as compared to their predecessors. Using software engineering practices, these complex systems can be developed in a systematic and efficient manner.

IEEE defines software engineering as "*the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*" In a nutshell, software engineering can be defined as the technological and managerial discipline concerned with systematic production and maintenance of software that is developed and modified on time and within cost estimates.

Software engineering is a discipline, which can be described as the combination of techniques of engineering and all aspects of software development. This includes design, implementation, and maintenance of software. It includes standardised approach to program development, both in its managerial and technical aspects.

The foundation for software engineering lies in the good working knowledge of computer science theory and practice. The theoretical background involves knowing how and when to use data structures, algorithms, and understanding what problems can be solved and what cannot. The practical knowledge includes through understanding of the workings of the hardware as well as thorough knowledge of the available programming languages and tools.

One of the main objectives of software engineering is to help developers obtain high-quality software. This quality is achieved through use of *Total Quality Management*, which enables continuous process improvement custom that leads to the development of more established approaches to software engineering.

Software Engineering Layers: Software engineering can be viewed as a layered technology. The various layers are listed below:

- The process layer is an adhesive that enables rational and timely development of computer software. Process defines an outline for a set of key process areas that must be acclaimed for effective delivery of software engineering technology.
- The method layer provides technical knowledge for developing software. This layer covers a broad array of tasks that include requirements, analysis, design, program construction, testing, and support phases of the software development.
- The tools layer provides computerised or semi-computerised support for the process and method layer. Sometimes tools are integrated in such a way that other tools can use information created by one tool. This multi usage is commonly referred to as **computer aided software engineering** (CASE). CASE combines software, hardware, and software

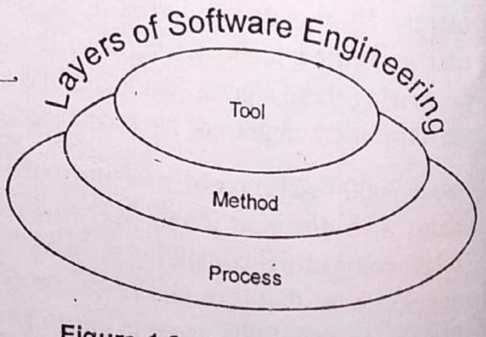


Figure 1.3 Layers of Software Engineering

engineering database to create software engineering analogous to computer-aided design (CAD) for hardware. CASE aids in application development including analysis, design, code generation, and debugging and testing. This is possible by using CASE tools, which provide automated methods for designing and documenting traditional-structure programming techniques. For example, the two prominent delivered technologies using CASE tools are application generators and PC-based workstations that provide graphics-oriented automation of the development process.

1.3.1 Software Engineer

A software engineer is an individual responsible for analysis, design, testing, implementation, and maintenance of effective and efficient software system. In addition, software engineer is also responsible for maintaining subsystems and external interfaces, subject to time and budgetary constraints.

Apart from management of analysis, specification, design and development of software applications, software engineers oversee the certification, maintenance, and testing of software applications. Software engineer also integrates the components of a complex software system. Generally, software engineer should possess the following qualities:

- **Problem solving skills:** Software engineer should develop algorithms and solve programming problems.
- **Programming skills:** Software engineer should be well versed in data structures and algorithms, and must be expert in one or more programming languages and possess strong programming capabilities.
- **Design approaches:** Software engineer should be familiar with numerous design approaches required during the development of software, at the same time, he should be able to translate ambiguous requirements and needs into precise specifications, and be able to converse with the use of a system in terms of applications.
- **Software technologies:** Software engineer should have good understanding of software technologies. Ability to move among several levels of abstractions at different stages of the software project, from specific application procedures and requirements to the detailed coding level is also required.
- **Project management:** Software engineer should know how to make a project work, on time and on budget, in order to produce quality applications and systems.
- **Model of the application:** Software engineer should be able to create and use a model of the application to guide choices of the many tradeoffs that will be faced by him. The model is used to find answers to questions about the behaviour of the system.

In addition to the above-mentioned qualities, software engineer should have good communication and interpersonal skills. Moreover knowledge of object-orientation, quality concept, International Organization of Standardization (ISO standards), and Capability Maturity Model (CMM) are also required. The tasks performed by software engineers have evolved rapidly, which has resulted in new areas of specialization and changing technology. Software engineers often work as part of a team that designs new hardware and software. This team comprises of engineering, marketing, manufacturing, and designing people who work together until the software is released.

10

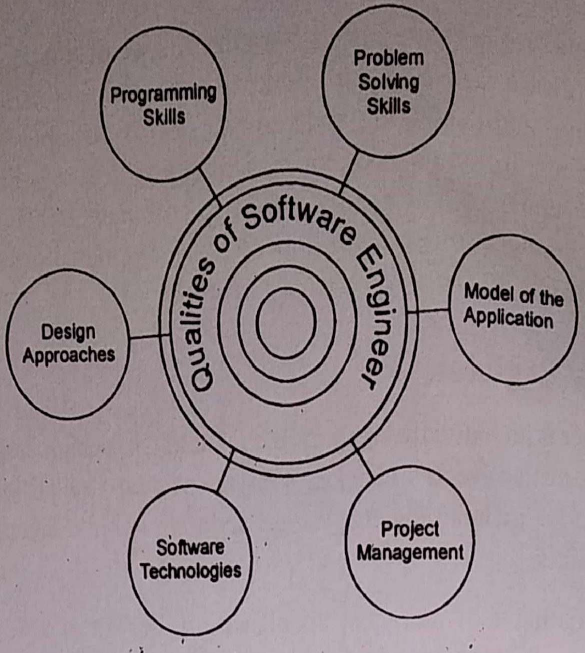


Figure 1.4 Skills of Software Engineer

1.4 PHASES IN SOFTWARE ENGINEERING

Software engineering shares common interest with other engineering disciplines. In the engineering domain, developing a solution to a given problem, whether building a bridge or making an electronic component, involves a sequence of interconnected steps. These steps or phases occur in software development as well. Also, since the prime objective of software engineering is to develop methods for large systems, which produce high quality software at low cost and in reasonable time, it is essential to perform software development in phases. This phased development of software is often referred to as **software development life cycle (SDLC)** or **software life cycle.**

A software development process comprises of different phases. These phases work in top to bottom approach, implying that the phases take inputs from the previous phases, add features, and then produce outputs. The outputs from different phases are referred to as **intermediate product, work product**, or derivable. The various phases involved in the systematic development of software are shown in Figure 1.5.

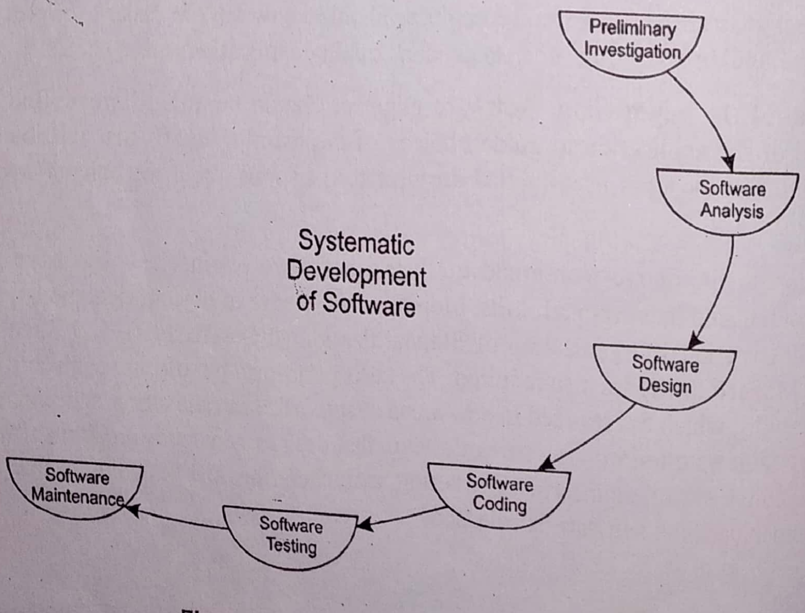


Figure 1.5 Software Development Process

gress

neering.

sibi-

gineer?

layer.

terial

1.4.1 Preliminary Investigation

This phase commences with discussion on the requests made by the user. The requests can be for a new system or modifying the existing system. An estimate is made of whether the identified user needs can be satisfied with the current hardware and software technologies or not. Preliminary investigation verifies the problem and understands the need for required system. It considers whether the proposed system will be cost effective from business point of view and whether it can be developed within existing budgetary constraints. In addition, time factor, which determines the duration of the project, is also considered.

Preliminary investigation should be quick and cost effective. The output of preliminary investigation decides whether the new system should be developed or not. There are three constraints, which decides the go or no-go decision.

- **Technical:** This evaluation determines whether technology needed for proposed system is available or not and if it is available then how can it be integrated within the organization. Technical evaluation also determines whether the existing system can be upgraded to use new technology and whether the organization has the expertise to use it or not.
- **Time:** This evaluation determines the time needed to complete a project. Time is an important issue in software development as cost increases with an increase in the time period of a project.
- **Budgetary:** This evaluation looks at the financial aspect of the project. Budgetary evaluation determines whether the investment needed to implement the system will be recovered at later stages or not.

(a) ***Software Analysis***: This phase studies the problem or requirements of software in detail. These requirements define the processes to be managed during the software development. After analysing the requirements of the user, a requirement statement known as **software requirement specification** (SRS) is developed. After analyses, planning for the project begins. It includes developing plans that describes the activities to be performed during the project, such as software configuration management plans, project and scheduling, and the quality assurance plans. In addition, the resources required during the project are also determined.

Table 1.2 Building Bridge and Corresponding SDLC Phase

Phase	Building Bridge	SDLC Phase
Formulate the problem by understanding the nature and general requirements of the problem.	Understand the load of the bridge it must carry, the approximate locations where it can be built, the height requirements, and so on.	Preliminary investigation.
Defining the problem precisely.	Specify the site for the bridge, its size, and a general outline of the type of bridge to be built.	Software requirement analysis and specifications.
Detailing the solution to the problem.	Determine exact configuration, size of the cables and beams, and developing blueprints for the bridge.	Software design.
Implementing.	Correspond to actual building of the bridge.	Software coding.
Checking.	Specify load, pressure, endurance, and robustness of the bridge.	Software testing.
Maintaining.	Specify repainting, repaving, and making any other repairs, which are necessary.	Software maintenance.

(b) Software Design: In this phase the requirements are given a 'defined' form. Design can be defined as a process of deciding information structures, in terms of efficiency, flexibility, and reusability. During this phase, strategic and tactical decisions are made to meet the required functional and quality requirements of a system. Software design serves as the blueprint for the implementation of requirement in the software system. Each element of the analysis model in the analysis phase provides information that is required to create design models. The requirement specification of software, together with data, functional, and behavioural models provides a platform to feed the design task to meet required functional and quality requirements of a system.

(c) Software Coding: This phase can be defined as a process of translating the software requirements into a programming language using tools that are available. Writing a software code requires a thorough knowledge of programming language and its tools. Therefore, it is important to choose the appropriate programming language according to the user requirements. A program code is efficient if it makes optimal use of resources and contains minimum errors.

Writing an efficient software code requires thorough knowledge of programming. However, to implement programming, *coding style* is followed. This style is used for writing software code in a programming language. Coding style also helps in writing the software code efficiently and with minimum errors. To ensure that all developers work in a harmonised manner (the source code should reflect a harmonised style, as if a single developer has written the entire code in one session), the developers should be aware of the coding guidelines before the inception of a software project.

(d) Software Testing: This testing is performed to assure that software is free from errors. Efficient testing improves the quality of software. To achieve this, software testing requires a thorough analysis of the software in a systematic manner. Test plan is created to test software in a planned and systematic manner. In addition, software testing is performed to ensure that software produces the correct outputs. This implies that outputs produced should be according to user requirements.

(e) Software Maintenance: This phase comprises of a set of software engineering activities that occur after software is delivered to the user. The objective of software maintenance is to make software operational according to user requirements. The need of software maintenance is to provide continuity of service. This implies that software maintenance focuses on fixing errors, recovering from failures, such as hardware failures, or incompatibility of hardware with software. In addition, it facilitates future maintenance work by modifying the software code and databases used in the software.

After the software is developed and delivered, it may require changes. Sometimes, changes are made in software system when user requirements are not completely met. To make changes in software system, software maintenance process evaluates, controls, and implements changes. Note that changes can also be forced on the software system because of changes in government regulations or changes in policies of the organization.

1.4.2 Case Study: Bridge and Software Development

Requirements analysis, design, and implementation are concerned with, what to do, how to do, and the way to do respectively. For example, Table 1.5 lists the phases involved in building a bridge and also lists the corresponding software development phase.

1.5 SOFTWARE PROJECT MANAGEMENT

Project management is concerned with the overall planning and coordination of a project from its commencement to its completion. This involves application of knowledge, skills, tools and techniques to meet the user's requirements within the specified time and cost. Effective software project management primarily concentrates on process, project, and product.

A process is defined as a series of steps involving activities and resources, which produce the desired output. Process can also be defined as a collection of procedures to develop a software product according to certain goals or standards. Generally, following points are noted about software processes:

- Process uses resources subject to given constraints, and produces intermediate and final products.
- Processes are composed of sub-processes that are organised in such a manner that each sub-process has its own process model.
- Each process is carried out with entry and exit criteria that help in monitoring the beginning and completion of the activity.
- Every process includes guidelines, which explain the objectives of each activity.
- Processes are vital because they impose uniformity on the set of activities.
- A process is regarded more than procedure, tools and techniques, which are collectively used in a structured manner to produce a product.
- Software processes include various technical and management issues, which are required to develop software.

The characteristics of software processes are listed in Table 1.6.

Table 1.3 Software Process Characteristics

Characteristics	Description
Understandability	The extent to which the process is explicitly defined and the ease with which the process definition is understood.
Visibility	Whether the process activities culminate in clear results or not so that the progress of the process is visible externally.
Supportability	The extent to which CASE tools can support the process activities.
Acceptability	The extent to which defined process is acceptable and usable by the engineers responsible for producing the software product.
Reliability	The manner in which the process is designed so that errors in the process are avoided or trapped before they result in errors in the product.
Robustness	Whether the process can continue inspite of unexpected problems or not.
Maintainability	Whether the process can evolve to reflect the changing organizational requirements or identify process improvements.
Rapidity	The speed with which the complete software can be delivered with given specifications.

A **project** is defined as a specification essential for developing or maintaining a specific product. A software project is developed when software processes or activities are executed for certain specific requirements of the user. Thus, using software process, software project can be easily developed. The activities in a software project comprises of various tasks for managing resources and developing product. Figure 1.6 shows that a software project involves people (developers, project manager, end users, and so on) also referred to as participants who use software processes to produce a product according to user requirements. The participants play a major role in the development of the project and they

N

Che

8. Def dev

9. Exp con ana inv

select the appropriate process for the project. In addition, a project is efficient if it is developed within the time constraint. The outcome or the result of a software project is known as **product**. *Thus, a software project uses software processes to produce a product.*

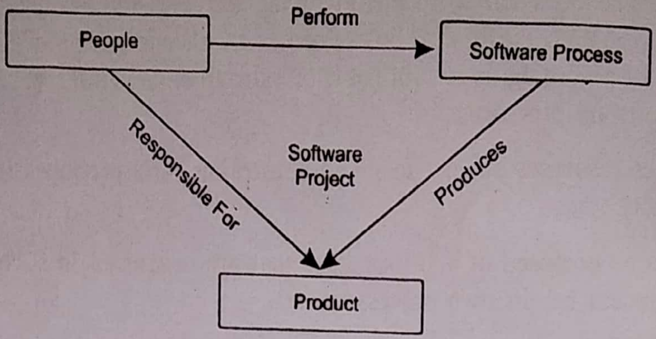


Figure 1.6 Software Project

Software process can consist of many software projects and each of them can produce one or more software products. The interrelationship between these three entities (process, project, and product) is shown in Figure 1.7. A software project begins with requirements and ends with the accomplishment of requirements. Thus, software process should be performed to develop final software by accomplishing the user requirements. Note that software processes are not specific to the software project.

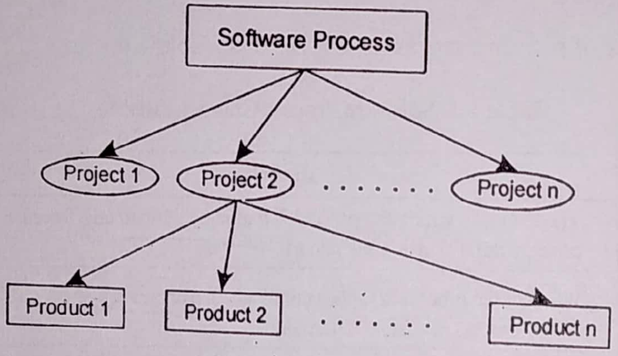


Figure 1.7 Processes, Projects, and Products

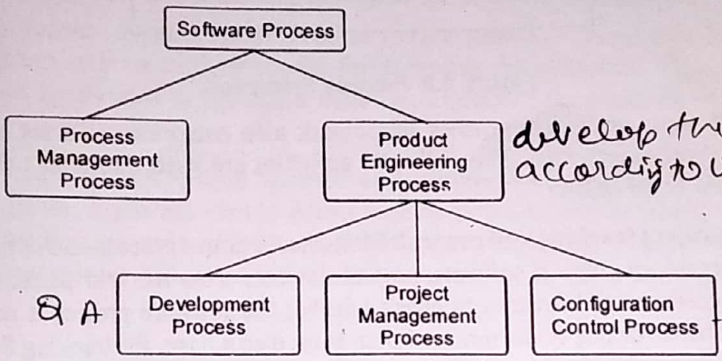
1.5.1 Process Management and Product Engineering Process

As stated above, the objective of software process is to develop a product, which accomplishes user requirements. For this, software processes requires components, which are shown in Figure 1.8. The major components of software process include process management process and product engineering process. The **process management processes** (PMP) aims at improving software processes so that a cost effective and a high-quality product is developed. To achieve a high quality product, the existing processes of the completed project are examined. The process of comprehending the existing process, analysing its properties, determining how to improve it, and then affecting the improvement is carried out by PMP. A group known as **software engineering process group** (SEPG) performs the activities of process management.

Based on the analysis stated above, the **product engineering processes** are improved, thereby improving the software process. The aim of product engineering processes is to develop the product according to user requirements. The product engineering process comprises of three major components, which are listed below:

- **Development process:** Implies the process, which is used during the development of software. It specifies the development and quality assurance activities that are to be performed. Programmers, designers, testing personnel, and others perform these processes.

- **Project management process:** Provides means to plan, organise and control the allocated resources to accomplish project costs, time and performance objectives. For this, various processes, techniques and tools are used to achieve the objectives of the project. Project management performs the activities of this process. Also, project management process is concerned with the set of activities or tasks, which are used to successfully accomplish a set of goals.
- **Configuration control process:** Manages changes that occur as a result of modifying the requirements. In addition, it maintains integrity of the products with the changed requirements. The activities in configuration control process are performed by a group called configuration control board (CCB).



Software Process
Process Management Process
Product Engineering Process
Development Process
Project Management Process
Configuration Control Process
t provide effective d high quality product
develop the product according to user requirement
QA
- Responsible for r
organized control the allocated resource

Figure 1.8 Software Processes

Note that project management process and configuration control process depend on the development process. The management process aims to control the development process, depending on the activities in the development process.

1.5.2 Process Framework

Process framework determines the processes, which are essential for completing a complex software project. This framework identifies certain activities, which are applicable to all software projects, regardless of their type and complexity. The activities used for these purposes are commonly referred to as **framework activities**, as shown in Figure 1.9. Some of the framework activities are listed below:

- **Communication:** Involves communication with the user so that the requirements are easy to understand.
- **Planning:** Establishes a project plan for the project. In addition, it describes the schedule for the project, technical tasks involved, expected risks and the required resources.
- **Modelling:** Encompasses creation of models, which allows the developer and the user to understand software requirements. In addition, it determines the designs to achieve those requirements.
- **Construction:** Combines generation of code with testing to uncover errors in the code.
- **Deployment:** Implies that the final product (that is, the software) is delivered to the user. The user evaluates the delivered product and provides a feedback based on the evaluation.

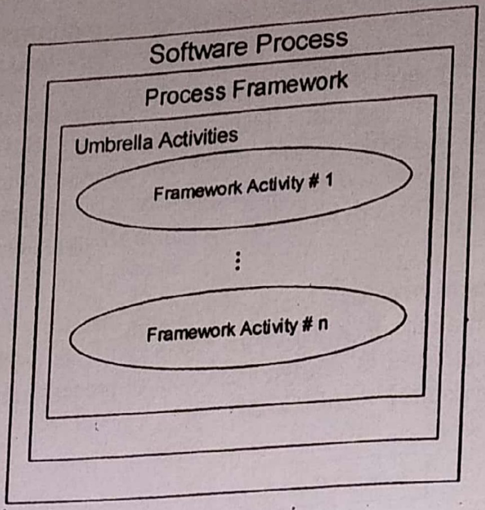


Figure 1.9 Process Framework

In addition to these activities, process framework also comprises of a set of activities known as ~~umbrella activities~~. The umbrella activities are used throughout the software process and are listed below:

- **Software project tracking and control:** Monitors the actual process so that management can take necessary steps if software project deviates from the laid plans. It involves tracking procedures and reviews to check whether the software project is according to user requirements or not. A documented plan is used as a basis for tracking the software activities and revising the plans. The management monitors these activities.
- **Formal technical reviews:** Assess the code, products and documents of software engineering practises to detect errors.
- **Software quality assurance:** Assures that software is according to the requirements. In addition, it is designed to evaluate the processes of developing and maintaining quality of the software.
- **Reusability management:** Determines the criteria for products' reuse and establishes mechanisms to achieve reusable components.
- **Software configuration management:** Manages the changes made in the software processes of the products throughout the software project life cycle. It controls changes made to the configuration and maintains the integrity in the software development process.
- **Risk management:** Identifies, analyses, evaluates and eliminates the possibility of unfavourable deviations from expected results, by a systematic activity and then develops strategies to manage them.

1.6 PROCESS MODELS

A process model also known as **software engineering paradigm** can be defined as a strategy which comprises of processes, methods, tools or steps for developing software. These models provide a basis for controlling various activities required to develop and maintain software. In addition, it helps the software development team in facilitating and understanding the activities involved in the project.

A process model for software engineering depends on the nature and application of software project. Thus, it is essential to define process models for each software project. IEEE defines process model as "*a framework containing the processes, activities, and tasks* *involved in the development, operation, and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use.*" Process model reflects the goals of software development, such as developing a high quality product and meeting the schedule on time. In addition, it provides a flexible framework for enhancing the processes. Other advantages of the software process model are listed below:

- **Enables effective communication:** Enhances understanding and provides a specific basis for process execution.
- **Facilitates process reuse:** Process development is a time-consuming and expensive activity, thus, software development team utilise the existing processes for different projects.
- **Effective:** Since process models can be used again and again; reusable processes provide an effective means for implementing processes for software development.
- **Facilitates process management:** Process models provide a framework for defining process status criteria and measures for software development. Thus, effective management is essential to provide a clear description of the plans for the software project.

Every software development process model takes requirements as input and delivers product as output. However, a process should detect defects in the phases in which they occur. This requires verification and validation (V&V) of the products after each and every phase of software development lifecycle.

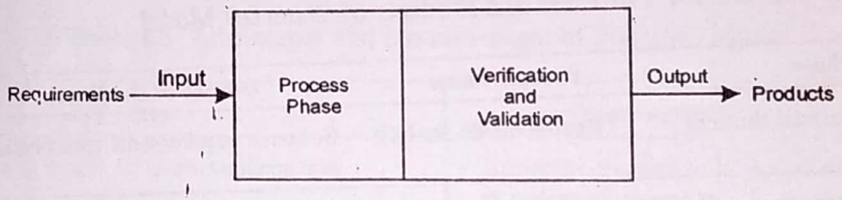


Figure 1.10 Phases in Development Process

Verification is the process of evaluating a system or its component for determining the product developed at each phase of software development. IEEE defines verification as "*a process for determining whether the software products of an activity fulfil the requirements or conditions imposed on them in the previous activities.*" Thus, it confirms that the product is transformed from one form to another as intended and with sufficient accuracy.

Validation is the process of evaluating the product at the end of each phase to ensure compliance with the requirements. In addition, it is the process of establishing a procedure and a method, which performs according to the intended outputs. IEEE defines validation as "*a process for determining whether the requirements and the final, as-built system or software product fulfils its specific intended use.*" Thus, validation substantiates the software functions with sufficient accuracy with respect to its requirement specifications.

Various kinds of process models used are *waterfall model, prototyping model, spiral model,* and *fourth generation techniques*.

1.6.1 Waterfall Model

In waterfall model (also known as **classical life cycle model**) the development of software proceeds linearly and sequentially from requirement analysis to design, coding, testing, integration, implementation, and maintenance. Thus, this model is also known as **linear sequential model.**

This model is simple to understand, and represents processes, which are easy to manage and measure. Waterfall model comprises of different phases and each phase has its distinct goal. Figure 1.11 shows that once a phase is completed, the development of software proceeds to the next phase. Each phase modifies the intermediate product to develop a new product as an output. The new product becomes the input of the next process as listed in Table 1.4.

NOTES

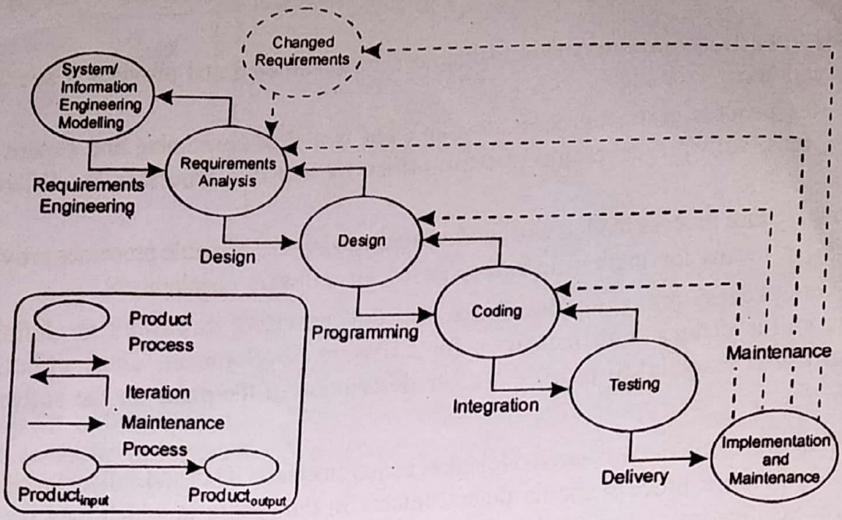


Figure 1.11 Waterfall Model

Table 1.4 Processes and Products of Waterfall Model

Input to the Phase	Process/Phase	Output of the Phase
Requirements defined through communication	Requirements analysis	Software requirements specification document
Software requirements specification document	Design	Design specification document
Design specification document	Coding	Executable software modules
Executable software modules	Testing	Integrated product
Integrated product	Implementation	Delivered software
Delivered software	Maintenance	Changed requirements

As stated earlier, waterfall model comprises of several phases. These phases are listed below:

- **System/information engineering modelling:** Establishes the requirements for the system known as **computer based system.** Hence, it is essential to establish the requirement of that system. A subset of requirements is allocated to the software. The system view is essential when the software interacts with the hardware. System engineering includes collecting requirements at the system level. The information gathering is necessary when the requirements are collected at a level where all decisions regarding business strategies are taken.
- **Requirement analysis:** Focuses on the requirements of the software which is to be developed. It determines the processes that are incorporated during the development of software. To specify the requirements' users specification should be clearly understood and the requirements should be analysed. This phase involves interaction between user and software engineer, and produces a document known as **software requirement specification** (SRS).
- **Design:** Determines the detailed process of developing software after the requirements are analysed. It utilises software requirements defined by the user and translates them

into a software representation. In this phase, the emphasis is on finding a solution to the problems defined in the requirement analysis phase. The software engineer, in this phase is mainly concerned with the data structure, algorithmic detail, and interface representations.

- **Coding:** Emphasises on translation of design into a programming language using the coding style and guidelines. The programs created should be easy to read and understand. All the programs written are documented according to the specification.
- **Testing:** Ensures that the product is developed according to the requirements of the user. Testing is performed to verify that the product is functioning efficiently with minimum errors. It focuses on the internal logics and external functions of the software and ensures that all the statements have been exercised (tested). Note that testing is a multi-stage activity, which emphasises verification and validation of the product.
- **Implementation and maintenance:** Delivers fully functioning operational software to the user. Once the software is accepted and deployed at the user's end, various changes occur due to changes in external environment (these include upgrading new operating system or addition of a new peripheral device). The changes also occur due to changing requirements of the user and the changes occurring in the field of technology. This phase focuses on modifying software, correcting errors, and improving the performance of the software.

The various advantages and disadvantages associated with waterfall model are listed in Table 1.5.

Table 1.5 Advantages and Disadvantages of Waterfall Model

Advantages	Disadvantages
▪ Relatively simple to understand. ▪ Each phase of development proceeds sequentially. ▪ Allows managerial control where a schedule with deadlines is set for each stage of development. ▪ Helps in controlling schedules, budgets, and documentation.	▪ Requirements need to be specified before the development proceeds. ▪ The changes of requirements in later phases of the waterfall model cannot be done. This implies that once an application is in the testing phase, it is difficult to incorporate changes at such a late phase. ▪ No user involvement and working version of the software is available when the software is developed. ▪ Does not involve risk management. ▪ Assumes that requirements are stable and are frozen across the project span.

1.6.2 Prototyping Model

The prototyping model is applied when there is an absence of detailed information regarding input and output requirements in the software. Prototyping model is developed on the assumption that it is often difficult to know all the requirements at the beginning of a project. It is usually used when there does not exist a system or in case of large and complex system where there is no manual process to determine the requirements.

Prototyping model increases flexibility of the development process by allowing the user to interact and experiment with a working representation of the product known as **prototype**. A prototype gives the user an actual feel of the system.

At any stage, if the user is not satisfied with the prototype, it can be thrown away and an entirely new system is developed. Generally, prototyping can be prepared by following the approaches listed below:

NOTES

- By creating major user interfaces without any substantive coding in the background in order to give the users a feel of what the system will look like.
- By abbreviating a version of the system that will perform limited subsets of functions.
- Using system components to demonstrate functions that will be included in the developed system.

Figure 1.12 illustrates the steps carried out in prototyping model. All these steps are listed below:

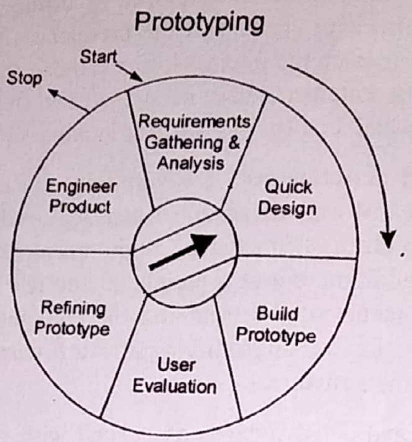


Figure 1.12 Prototyping Model

1. **Requirements gathering and analysis:** Prototyping model begins with requirements analysis, and the requirements of the system are defined in detail. The user is interviewed to know the requirements of the system.
2. **Quick design:** When requirements are known, a preliminary design or a quick design for the system is created. It is not a detailed design, however, it includes the important aspects of the system, which gives an idea of the system to the user. Quick design helps in developing the prototype.
3. **Build prototype:** Information gathered from quick design is modified to form a prototype. The first prototype of the required system is developed from quick design. It represents a 'rough' design of the required system.
4. **Assessment or user evaluation:** Next, the proposed system is presented to the user for consideration as a part of development process. The users thoroughly evaluate the prototype and recognise its strengths and weaknesses, such as what is to be added or removed. Comments and suggestions are collected from the users and are provided to the developer.
5. **Prototype refinement:** Once the user evaluates the prototype, it is refined according to the requirements. The developer revises the prototype to make it more effective and efficient according to the user requirement. If the user is not satisfied with the developed prototype, then a new prototype is developed with the additional information provided by the user. The new prototype is evaluated in the same manner, as the previous prototype. This process continues until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the final prototype.
6. **Engineer product:** Once the requirements are completely known, user accepts the final prototype. The final system is thoroughly evaluated and tested followed by routine maintenance on continuing basis to prevent large-scale failures and to minimise downtime.

The various advantages and disadvantages associated with prototyping model are listed in Table 1.6.

NOTES

Table 1.6 Advantages and Disadvantages of Prototyping Model

Advantages	Disadvantages
▪ Provides a working model to the user early in the process, enabling early assessment and increasing user confidence. ▪ Developer gains experience and insight by developing a prototype, thereby resulting in better implementation of requirements. ▪ Prototyping model serves to clarify requirements, which are not clear, hence reducing ambiguity and improving communication between developer and user. ▪ There is a great involvement of users in software development. Hence, the requirements of the users are met to the greatest extent. ▪ Helps in reducing risks associated with the project.	▪ If the user is not satisfied by the developed prototype, then a new prototype is developed. This process goes on until a perfect prototype is developed. Thus, this model is time consuming and expensive. ▪ Developer looses focus of the real purpose of prototype and compromise with the quality of the product. For example, they apply some of the inefficient algorithms or inappropriate programming languages used in developing the prototype. ▪ Prototyping can lead to false expectations. It often creates a situation where user believes that the development of the system is finished when it is not. ▪ The primary goal of prototyping is rapid development, thus, the design of system can suffer as it is built in a series of layers without considering integration of all the other components.

1.6.3 Spiral Model

In 1980's Boehm introduced a process model known as **spiral model.** The spiral model comprises of activities organised in a spiral, which has many cycles. This model combines the features of prototyping model and waterfall model and is advantageous for large, complex and expensive projects. The spiral model determines requirement problems in developing the prototypes. In addition, spiral model guides and measures the need of risk management in each cycle of the spiral model. IEEE defines spiral model as *"a model of the software development process in which the constituent activities, typically requirements analysis, preliminary and detailed design, coding, integration, and testing, are performed iteratively until the software is complete."*

The objective of spiral model is to emphasise management to evaluate and resolve risks in the software project. Different areas of risks in the software project are project overruns, changed requirements, loss of key project personnel, delay of necessary hardware, competition from other software developers, and technological breakthroughs, which obsolete the project. Figure 1.13 shows the spiral model and the steps involved in the model are listed below:

1. Each cycle of the first quadrant commences with identifying the goals for that cycle. In addition, it determines other alternatives, which are possible in accomplishing those goals.
2. The next step in the cycle evaluates alternatives based on objectives and constraints. This process identifies the areas of uncertainty and focuses on significant sources of the project risks. Risk signifies that there is a possibility that the objectives of the project cannot be accomplished. If so the formulation of a cost effective strategy for resolving risks is followed. Figure 1.13 shows the strategy, which includes prototyping, simulation, benchmarking administrating user questionnaires or risk resolution technique.
3. The development of the software depends on remaining risks. The third quadrant develops the final software while considering the risks that can occur. Risk management considers the time and effort to be devoted to each project activity, such as planning, configuration management, quality assurance, verification, and testing.

4. The last quadrant plans the next step, and includes planning for the next prototype and thus, comprises of requirements plan, development plan, integration plan, and test plan.

NOTES

One of the key features of the spiral model is that each cycle is completed by a review conducted by the individuals or users. This includes the review of all the intermediate products, which are developed during the cycles. In addition, it includes the plan for next cycle and the resources required for the cycle.

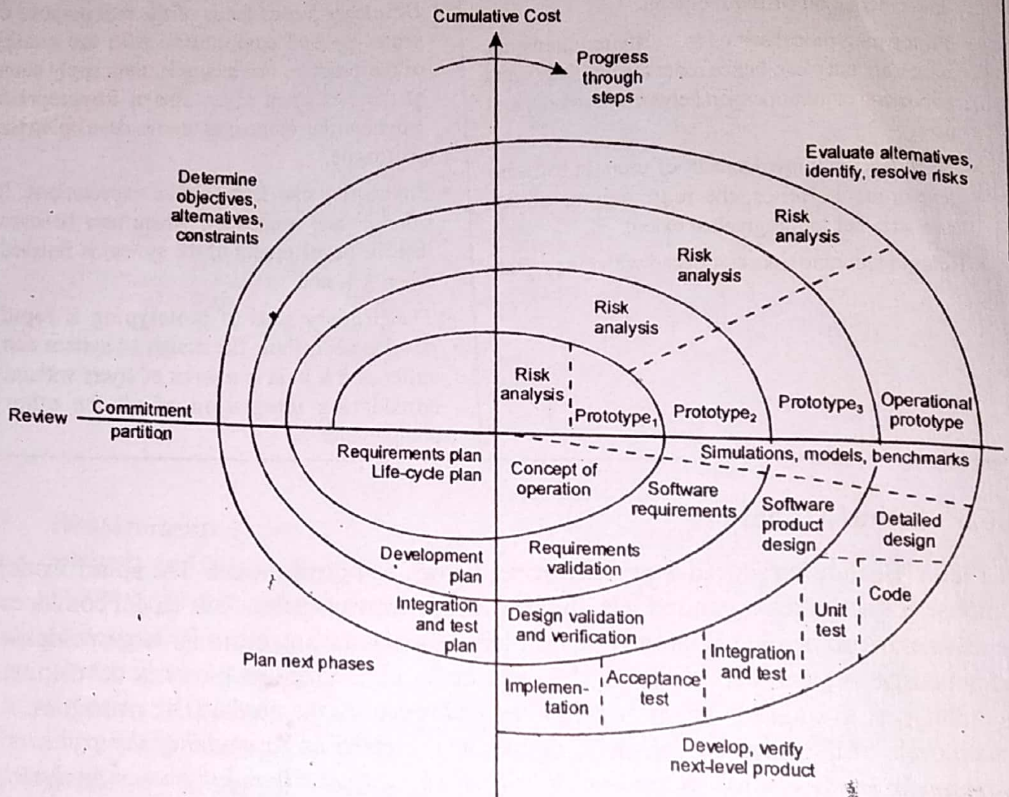


Figure 1.13 Spiral Model

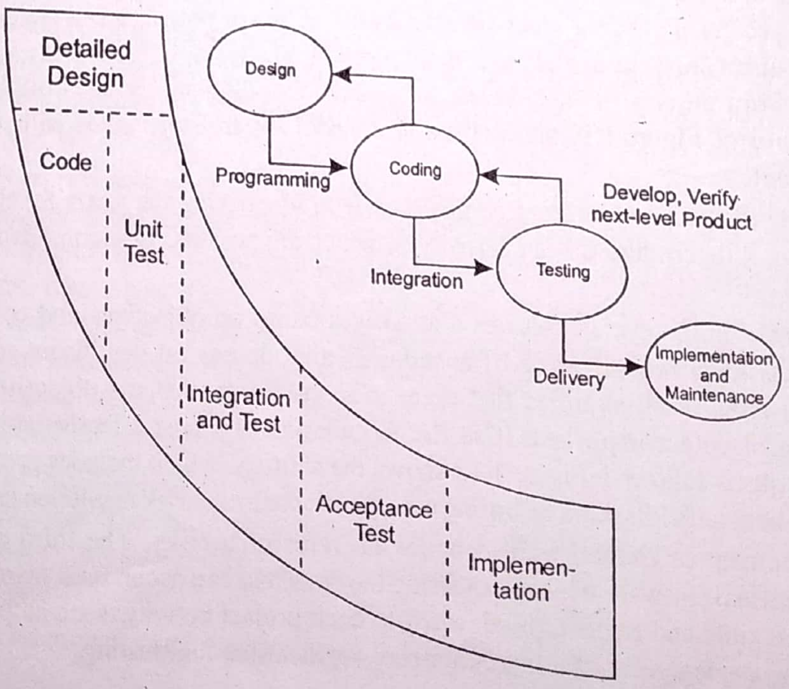


Figure 1.14 Spiral and Waterfall Model

The spiral model is similar to the waterfall model, as software requirements are understood at the early stages in both the models. However, the major risks involved with developing the final software are resolved in spiral model. When these issues are resolved, a detail design of the software is developed. Note that processes in the waterfall model are followed by different cycles in the spiral model as shown in Figure 1.14.

The various advantages and disadvantages associated with spiral model are listed in Table 1.7.

Spiral model is also similar to prototyping process model. As one of the key features of prototyping is to develop a prototype until the user requirements are accomplished. The second step of the spiral model functions similarly. The prototype is developed to clearly understand and achieve user requirements. If the user is not satisfied with the prototype, a new prototype known as *operational prototype* is developed.

Table 1.7 Advantages and Disadvantages of Spiral Model

Advantages	Disadvantages
▪ Avoids the problems resulting in risk-driven approach in the software. ▪ Specifies a mechanism for software quality assurance activities. ▪ Spiral model is utilised by complex and dynamic projects. ▪ Re-evaluation after each step allows changes in user perspectives, technology advances or financial perspectives. ▪ Estimation of budget and schedule gets realistic as the work progresses.	▪ Assessment of project risks and its resolution is not an easy task. ▪ Difficult to estimate budget and schedule in the beginning, as some of the analysis is not done until the design of the software is developed.

1.6.4 Fourth Generation Techniques (4GT)

Fourth generation techniques enable software engineers to specify characteristics of software at a high level and then automatically generate the source code. In addition to being a process model, fourth generation techniques are a collection of software tools used by software engineers to solve a problem by using a specialised language or a graphic notation so that users easily understand the problem. Hence, fourth generation techniques use instructions similar to spoken languages to allow the programmers to define *what* they want the computer to do rather than *how* to do it. For this, fourth generation techniques use certain tools, which are listed below:

- Non-procedural languages for database query.
- Report generation.
- Data manipulation.
- Screen interaction and definition.
- Code generation.
- High-level graphics capability.
- Spreadsheet capability.
- Automated generation of hypertext markup language and similar languages used for web-site creation using advanced software tools.

The various advantages and disadvantages associated with fourth generation techniques are listed in Table 1.8.

Table 1.8 Advantages and disadvantages of Fourth generation Techniques

Advantages	Disadvantages
▪ Development time is reduced, when used for small and intermediate applications. ▪ The interaction between user and developer helps in detection of errors. ▪ When integrated with CASE tools and code generators, fourth generation techniques provide a solution to most of the software engineering problems.	▪ Difficult to use. ▪ Limited only to small business information systems.

1.7 ROLE OF SOFTWARE METRICS AND MEASUREMENT

To achieve accurate schedule and cost estimate, better quality products, and higher productivity an effective software management is required, which in-turn can be attained through use of software metrics. A metric is a derived unit of measurement that cannot be directly observed, but is created by combining or relating two or more measures.

1.7.1 Software Measurement

To assess the quality of the engineered product or system and to better understand the models that are created, some measures are used. These measures are collected throughout the software development life cycle with an intention to improve the software **process** on a continuous basis. Measurement helps in estimation, quality control, productivity assessment, and project control throughout a software **project**. Also, measurement is used by software engineers to gain insight into the design and development of the work **products**. In addition, measurement assists in strategic decision-making as a project proceeds.

Software measurements are of two categories namely, direct measures and indirect measures. **Direct measures** include software processes like cost and effort applied and product like lines of code produced, execution speed, and other defects that have been reported. **Indirect measures** include products like functionality, quality, complexity, reliability, maintainability, and much more.

Generally, software measurement is considered as a management tool, which if conducted in an effective manner helps project manager and the entire software team to take decisions that lead to successful completion of the project. Measurement process is characterised by a set of five activities, which are listed below:

- **Formulation:** Performs measurement and develops appropriate metrics for software under consideration.
- **Collection:** Collects data to derive the formulated metrics.
- **Analysis:** Calculates metrics and use mathematical tools.
- **Interpretation:** Analyses the metrics to attain insight into the quality of representation.
- **Feedback:** Communicates recommendation derived from product metrics to the software team.

Note that collection and analysis activities drive the measurement process. In order to perform these activities effectively, it is recommended to automate data collection and analysis, establish guidelines and recommendations for each metric, and use statistical techniques to interrelate external quality features and internal product attributes.