Unit 3

Software Engineering
Software Engineering

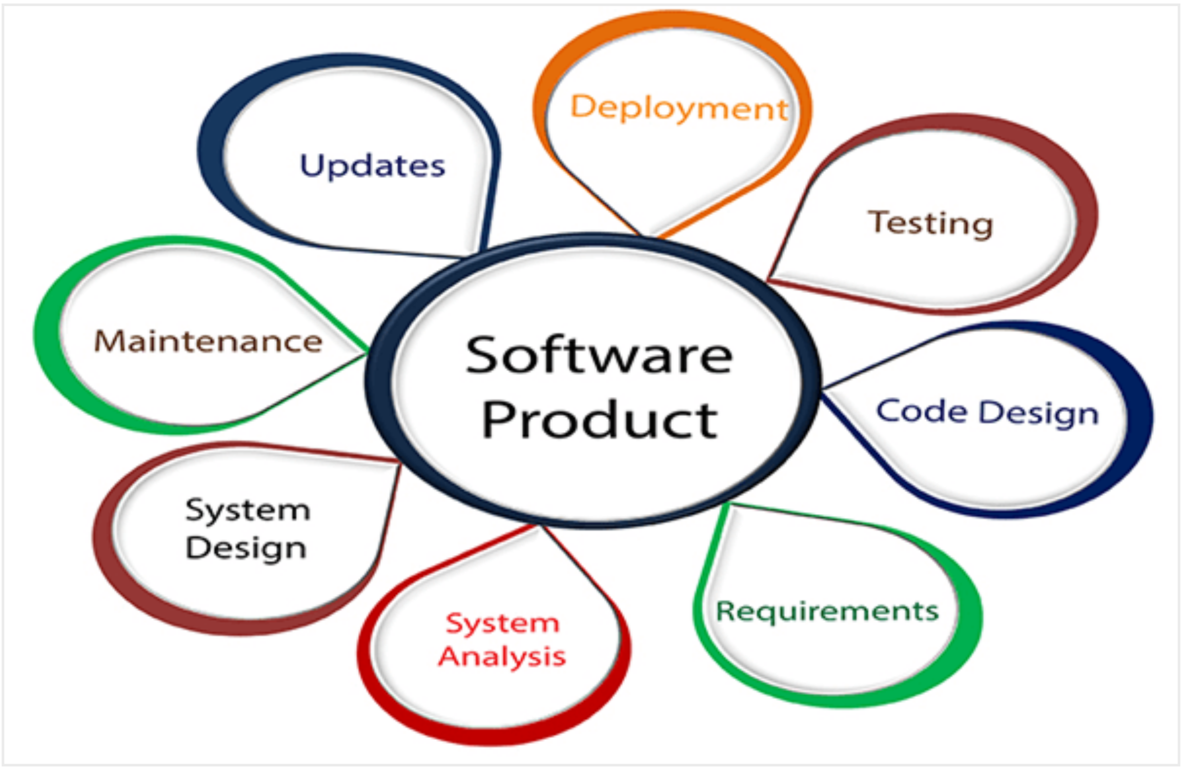
The term **software engineering** is the product of two words, **software**, and **engineering**.

The **software** is a collection of integrated programs.

Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.

Computer programs and related documentation such as requirements, design models and user manuals.

Engineering is the application of **scientific** and **practical** knowledge to **invent, design, build, maintain**, and **improve frameworks, processes, etc**.



Software Engineering is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

Need of Software Engineering

The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.

o **Huge Programming:** It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become

extensive engineering has to step to give it a scientific process.
o **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
o **Cost:** As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the proper process is not adapted.
o **Dynamic Nature:** The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.
o **Quality Management:** Better procedure of software development provides a better and quality software product.

Main Attributes of Software Engineering

Software Engineering is a systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance of a software system. There are four main Attributes of Software Engineering.

1. **Efficiency:** It provides a measure of the resource requirement of a software product efficiently.
2. **Reliability:** It assures that the product will deliver the same results when used in similar working environment.
3. **Reusability:** This attribute makes sure that the module can be used in multiple applications.
4. **Maintainability:** It is the ability of the software to be modified, repaired, or enhanced easily with changing requirements.

Dual Role of Software

There is a dual role of software in the industry. The first one is as a product and the other one is as a vehicle for delivering the product. We will discuss both of them.

1. As a Product

• It delivers computing potential across networks of Hardware.
• It enables the Hardware to deliver the expected functionality.
• It acts as an information transformer because it produces, manages, acquires, modifies, displays, or transmits information.

2. As a Vehicle for Delivering a Product

• It provides system functionality (e.g., payroll system).
• It controls other software (e.g., an operating system).
• It helps build other software (e.g., software tools).

Objectives of Software Engineering

1. **Maintainability:** It should be feasible for the software to evolve to

meet changing requirements.
2. **Efficiency:** The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
3. **Correctness:** A software product is correct if the different requirements specified in the SRS Document have been correctly implemented.
4. **Reusability:** A software product has good reusability if the different modules of the product can easily be reused to develop new products.
5. **Testability:** Here software facilitates both the establishment of test criteria and the evaluation of the software concerning those criteria.
6. **Reliability:** It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
7. **Portability:** In this case, the software can be transferred from one computer system or environment to another.
8. **Adaptability:** In this case, the software allows differing system constraints and the user needs to be satisfied by making changes to the software.
9. **Interoperability:** Capability of 2 or more functional units to process data cooperatively.

Program vs Software Product

Parameters	Program	Software Product
Definition	A program is a set of instructions that are given to a computer in order to achieve a specific task.	Software is when a program is made available for commercial business and is properly documented along with its licensing. Software Product = Program + Documentation + Licensing.

Stages Involved	Program is one of the stages involved in the development of the software.	Software Development usually follows a life cycle, which involves the feasibility study of the project, requirement gathering, development of a prototype, system design, coding, and testing.

Advantages of Software Engineering

There are several advantages to using a systematic and disciplined approach to software development, such as:

1. **Improved Quality:** By following established software engineering principles and techniques, the software can be developed with fewer bugs and higher reliability.
2. **Increased Productivity:** Using modern tools and methodologies can streamline the development process, allowing developers to be more productive and complete projects faster.
3. **Better Maintainability:** Software that is designed and developed using sound software engineering practices is easier to maintain and update over time.
4. **Reduced Costs:** By identifying and addressing potential problems early in the development process, software engineering can help to reduce the cost of fixing bugs and adding new features later on.
5. I**ncreased Customer Satisfaction:** By involving customers in the development process and developing software that meets their needs, software engineering can help to increase customer satisfaction.
6. **Better Team Collaboration:** By using Agile methodologies and continuous integration, software engineering allows for better collaboration among development teams.
7. **Better Scalability**: By designing software with scalability in mind, software engineering can help to ensure that software can handle an increasing number of users and transactions.
8. **Better Security:** By following the Software Development Life Cycle (SDLC) and performing security testing, software engineering can help to prevent security breaches and protect sensitive data.

In summary, software engineering offers a structured and efficient approach to software development, which can lead to higher-quality software that is easier to maintain and adapt to changing requirements. This can help to improve customer satisfaction and reduce costs, while also promoting better collaboration among development teams.

Disadvantages of Software Engineering

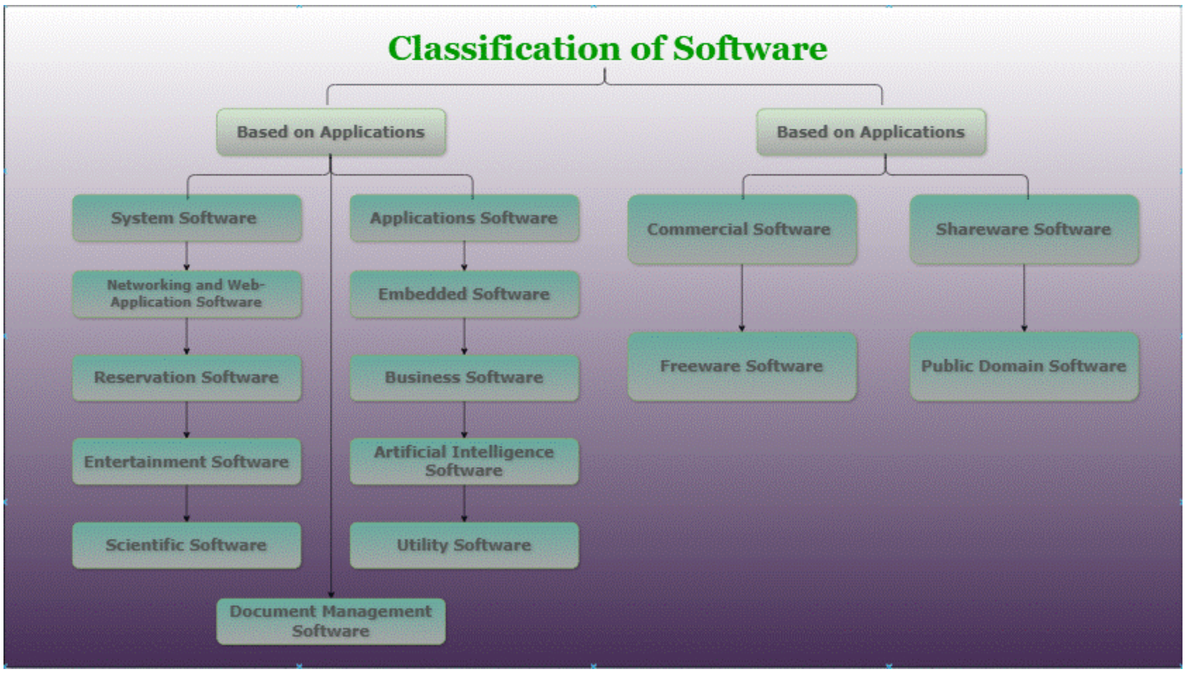
While Software Engineering offers many advantages, there are also some potential disadvantages to consider:

1. **High upfront costs:** Implementing a systematic and disciplined approach to software development can be resource-intensive and require a significant investment in tools and training.
2. **Limited flexibility:** Following established software engineering principles and methodologies can be rigid and may limit the ability to quickly adapt to changing requirements.
3. **Bureaucratic**: Software Engineering can create an environment that is bureaucratic, with a lot of processes and paperwork, which may slow down the development process.
4. **Complexity**: With the increase in the number of tools and methodologies, software engineering can be complex and difficult to navigate.
5. **Limited creativity:** The focus on structure and process can stifle creativity and innovation among developers.
6. **High learning curve:** The development process can be complex, and it requires a lot of learning and training, which can be challenging for new developers.
7. **High dependence on tools:** Software engineering heavily depends on the tools, and if the tools are not properly configured or are not compatible with the software, it can cause issues.
8. **High maintenance**: The software engineering process requires regular maintenance to ensure that the software is running efficiently, which can be costly and time-consuming.

Types of Software

The software is used extensively in several domains including hospitals, banks, schools, defense, finance, stock markets, and so on.

It can be categorized into different types:



Classification of Software

1. **Based on Application**
2. **Based on Copyright**

1. Based on Application

The software can be classified on the basis of the application. These are to be done on this basis.

1. System Software:

System Software is necessary to manage computer resources and support the execution of application programs. Software like operating systems, compilers, editors and drivers, etc., come under this category. A computer cannot function without the presence of these. Operating systems are needed to link the machine-dependent needs of a program with the capabilities of the machine on which it runs. Compilers translate programs from high-level language to machine language.

2. Application Software:

Application software is designed to fulfill the user's requirement by interacting with the user directly. It could be classified into two major categories:- generic or customized. Generic Software is software that is open to all and behaves the same for all of its users. Its function is limited and not customized as per the user's changing requirements. However, on the other hand, customized software is the software products designed per the client's requirement, and are not available for all.

3. Networking and Web Applications Software:

Networking Software provides the required support necessary for computers to interact with each other and with data storage facilities.

Networking software is also used when software is running on a network of computers (such as the World Wide Web). It includes all network management software, server software, security and encryption software, and software to develop web-based applications like HTML, PHP, XML, etc.

4. Embedded Software:

This type of software is embedded into the hardware normally in the Read-Only Memory (ROM) as a part of a large system and is used to support certain functionality under the control conditions. Examples are software used in instrumentation and control applications like washing machines, satellites, microwaves, etc.

5. Reservation Software:

A Reservation system is primarily used to store and retrieve information and perform transactions related to air travel, car rental, hotels, or other activities. They also provide access to bus and railway reservations, although these are not always integrated with the main system. These are also used to relay computerized information for users in the hotel industry, making a reservation and ensuring that the hotel is not overbooked.

6. Business Software:

This category of software is used to support business applications and is the most widely used category of software. Examples are software for inventory management, accounts, banking, hospitals, schools, stock markets, etc.

7. Entertainment Software:

Education and Entertainment software provides a powerful tool for educational agencies, especially those that deal with educating young children. There is a wide range of entertainment software such as computer games, educational games, translation software, mapping software, etc.

8. Artificial Intelligence Software:

Software like expert systems, decision support systems, pattern recognition software, artificial neural networks, etc. come under this category. They involve complex problems which are not affected by complex computations using non-numerical algorithms.

9. Scientific Software:

Scientific and engineering software satisfies the needs of a scientific or engineering user to perform enterprise-specific tasks. Such software is written for specific applications using principles, techniques, and formulae particular to that field. Examples are software like MATLAB, AUTOCAD, PSPICE, ORCAD, etc.

10. Utility Software:

The programs coming under this category perform specific tasks and are different from other software in terms of size, cost, and complexity. Examples are antivirus software, voice recognition software, compression programs, etc.

11. Document Management Software:

Document Management Software is used to track, manage, and store documents to reduce the paperwork. Such systems are capable of keeping a record of the various versions created and modified by different users (history tracking). They commonly provide storage, versioning, metadata, security, as well as indexing and retrieval capabilities.

2. Based on Copyright

Classification of Software can be done based on copyright. These are stated as follows:

1. Commercial Software:

It represents the majority of software that we purchase from software companies, commercial computer stores, etc. In this case, when a user buys software, they acquire a license key to use it. Users are not allowed to make copies of the software. The company owns the copyright of the program.

2. Shareware Software:

Shareware software is also covered under copyright, but the purchasers are allowed to make and distribute copies with the condition that after testing the software, if the purchaser adopts it for use, then they must pay for it. In both of the above types of software, changes to the software are not allowed.

3. Freeware Software:

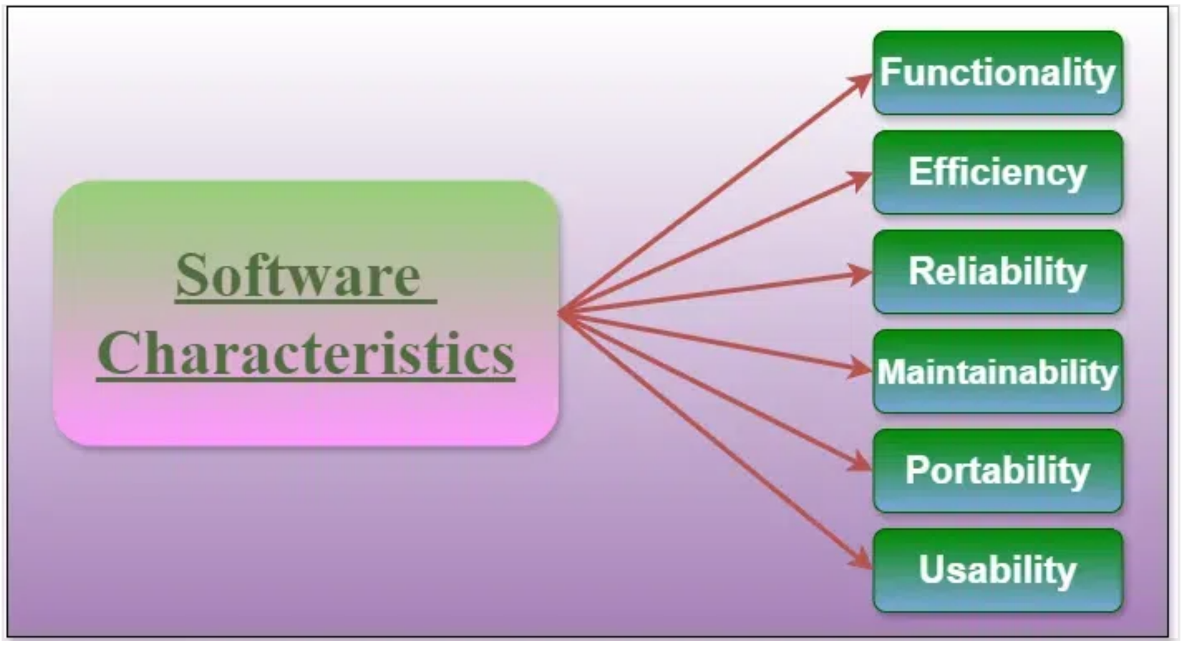
In general, according to freeware software licenses, copies of the software can be made both for archival and distribution purposes, but here, distribution cannot be for making a profit. Derivative works and modifications to the software are allowed and encouraged. Decompiling of the program code is also allowed without the explicit permission of the copyright holder.

4. Public Domain Software:

In the case of public domain software, the original copyright holder explicitly relinquishes all rights to the software. Hence, software copies can be made both for archival and distribution purposes with no restrictions on distribution. Modifications to the software and reverse engineering are also allowed.

Software Characteristics

There are 6 components of Software Characteristics are discussed here. We will discuss each one of them in detail.



Software Characteristics

Functionality:

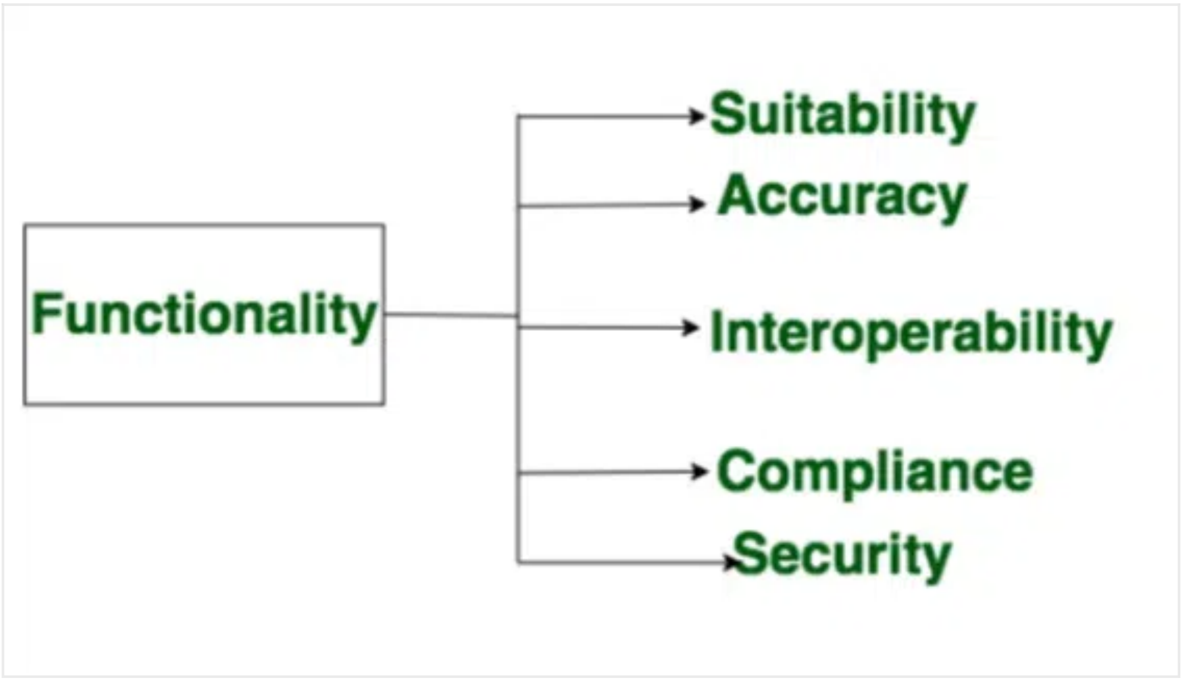
It refers to the degree of performance of the software against its intended purpose.

Functionality refers to the set of features and capabilities that a software program or system provides to its users. It is one of the most important characteristics of software, as it determines the usefulness of the software for the intended purpose. Examples of functionality in software include:

- Data storage and retrieval
- Data processing and manipulation
- User interface and navigation
- Communication and networking
- Security and access control
- Reporting and visualization
- Automation and scripting

The more functionality a software has, the more powerful and versatile it is, but also the more complex it can be. It is important to balance the need for functionality with the need for ease of use, maintainability, and scalability.

Required functions are:



Functionality

Reliability:

A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time.

Reliability is a characteristic of software that refers to its ability to perform its intended functions correctly and consistently over time. Reliability is an important aspect of software quality, as it helps ensure that the software will work correctly and not fail unexpectedly.

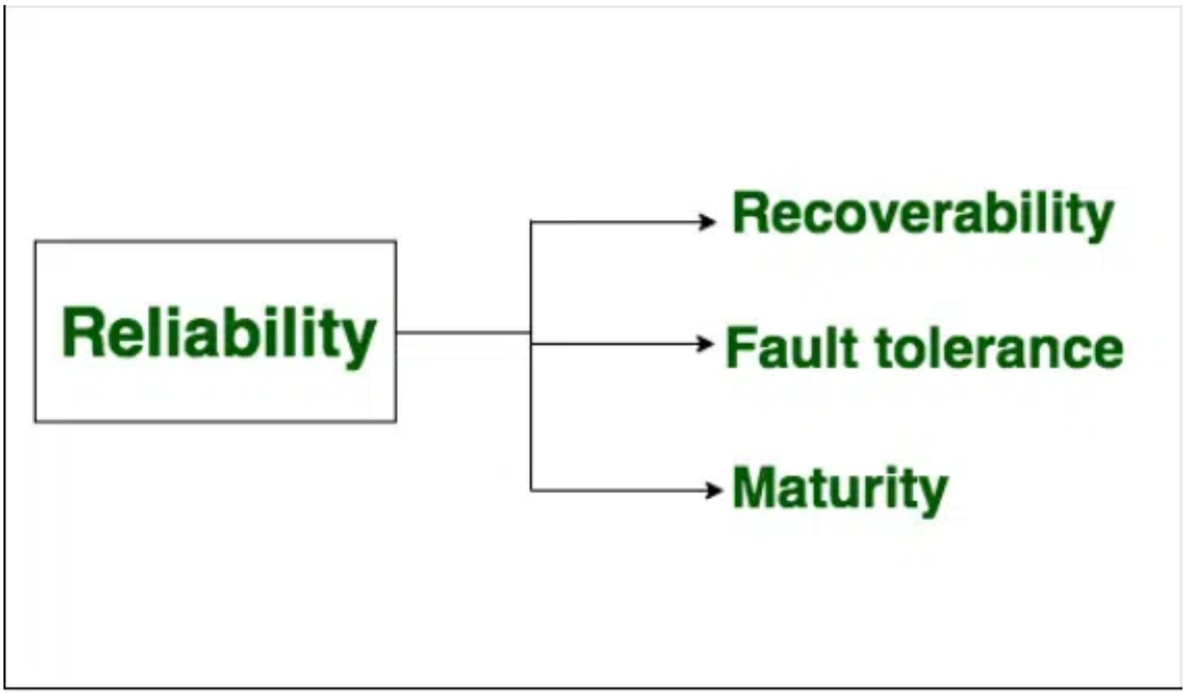
Examples of factors that can affect the reliability of software include:

1. Bugs and errors in the code
2. Lack of testing and validation
3. Poorly designed algorithms and data structures
4. Inadequate error handling and recovery
5. Incompatibilities with other software or hardware

To improve the reliability of software, various techniques, and methodologies can be used, such as testing and validation, formal verification, and fault tolerance.

Software is considered reliable when the probability of it failing is low and it is able to recover from the failure quickly, if any.

Required functions are:



Reliability

Efficiency:

It refers to the ability of the software to use system resources in the most effective and efficient manner. The software should make effective use of storage space and executive command as per desired timing requirements.

Efficiency is a characteristic of software that refers to its ability to use resources such as memory, processing power, and network bandwidth in an optimal way. High efficiency means that a software program can perform its intended functions quickly and with minimal use of resources, while low efficiency means that a software program may be slow or consume excessive resources.

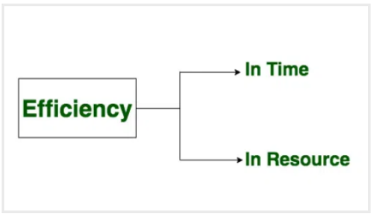
Examples of factors that can affect the efficiency of the software include:

1. Poorly designed algorithms and data structures
2. Inefficient use of memory and processing power
3. High network latency or bandwidth usage
4. Unnecessary processing or computation
5. Unoptimized code

To improve the efficiency of software, various techniques, and methodologies can be used, such as performance analysis, optimization, and profiling.

Efficiency is important in software systems that are resource-constrained, high-performance, and real-time systems. It is also important in systems that need to handle many users or transactions simultaneously.

Required functions are:

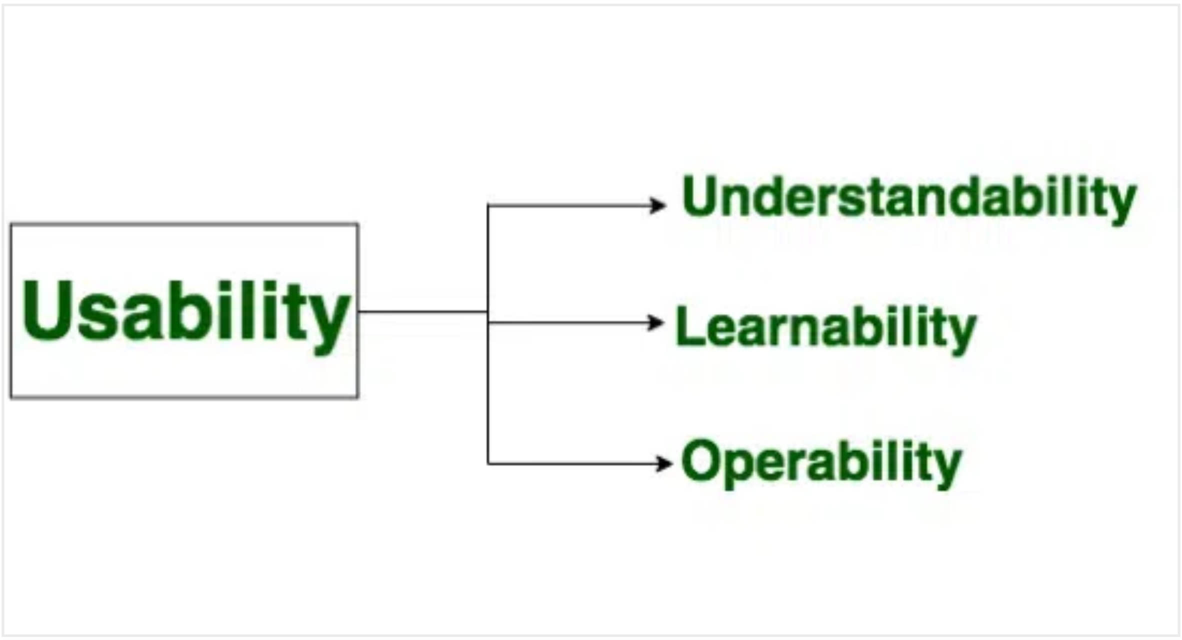


Efficiency

Usability:

It refers to the extent to which the software can be used with ease. the amount of effort or time required to learn how to use the software.

Required functions are:

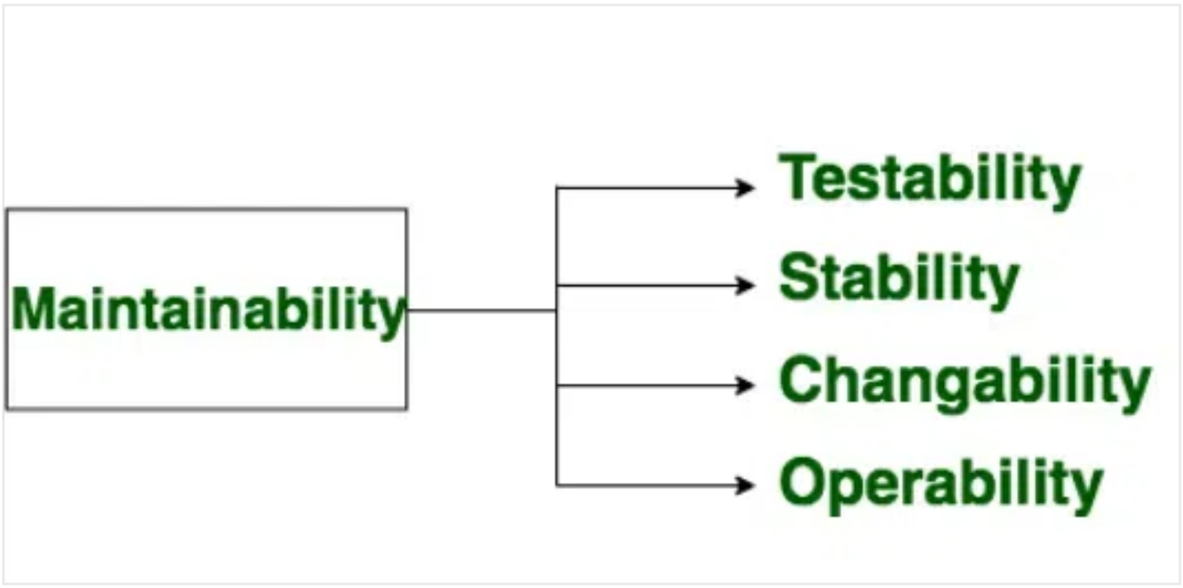


Usability

Maintainability:

It refers to the ease with which modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.

Required functions are:

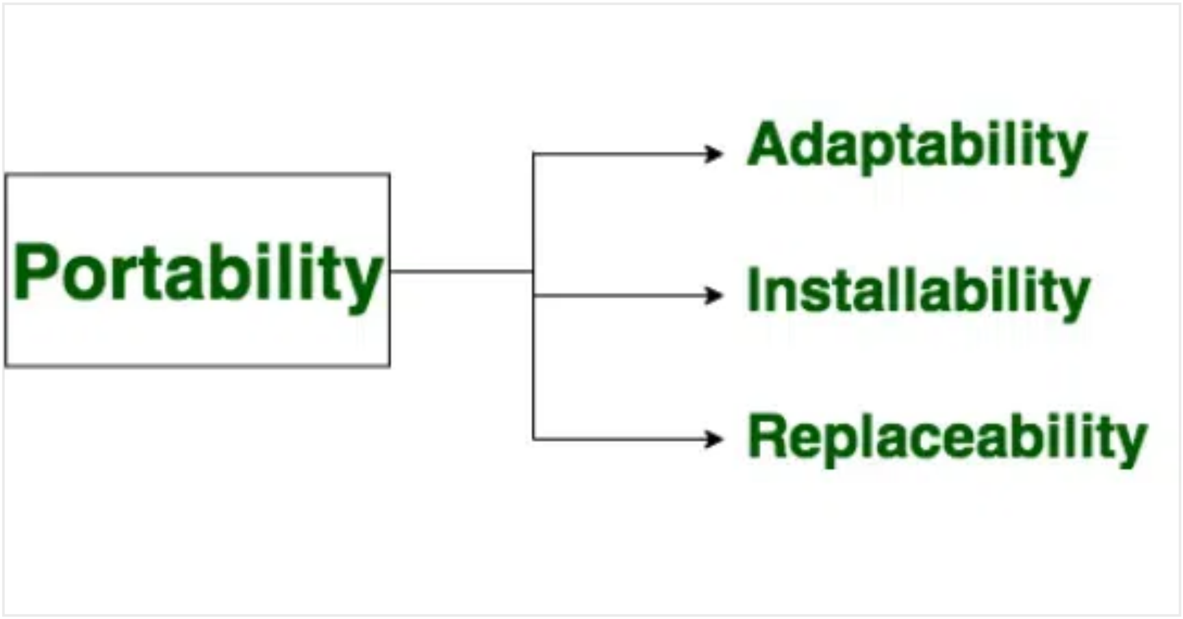


Maintainability

Portability:

A set of attributes that bears on the ability of software to be transferred from one environment to another, without minimum changes.

Required functions are:



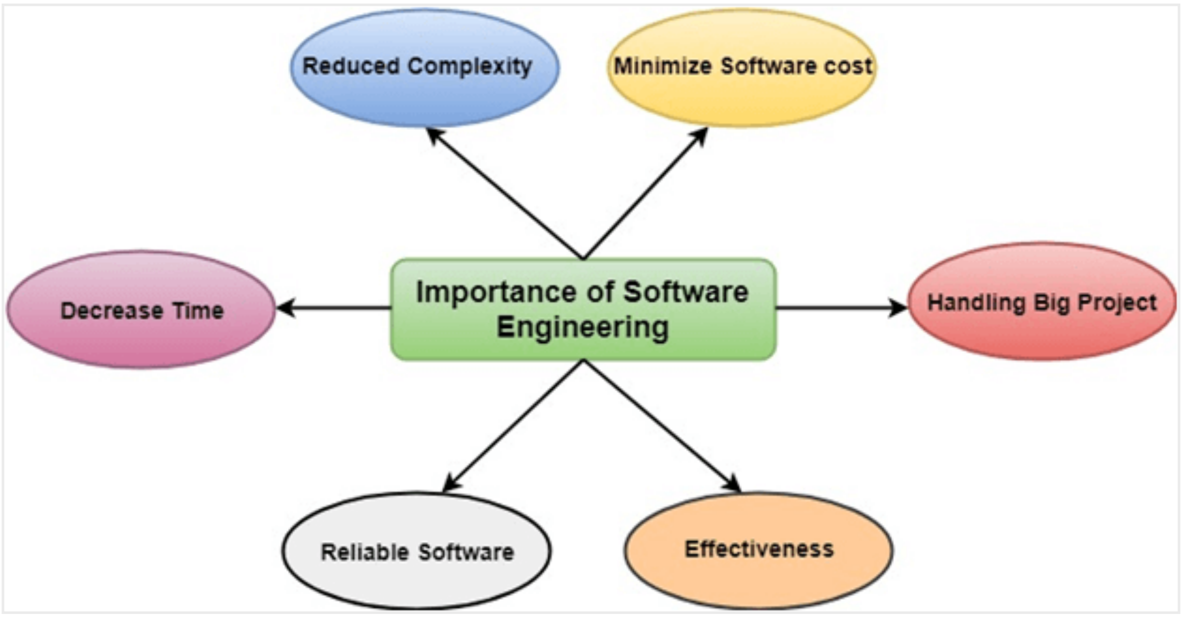
Characteristics of a good software engineer

The features that good software engineers should possess are as follows:

• Exposure to systematic methods, i.e., familiarity with software engineering principles.
• Good technical knowledge of the project range (Domain knowledge).
• Good programming abilities.

• Good communication skills. These skills comprise of oral, written, and interpersonal skills.
• High motivation.
• Sound knowledge of fundamentals of computer science.
• Intelligence.
• Ability to work in a team
• Discipline, etc.

Importance of Software Engineering



The importance of Software engineering is as follows:

1. **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.
2. **To minimize software cost:** Software needs a lot of hardwork and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.
3. **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So if you are making your software according to the software engineering method, then it will

decrease a lot of time.
4. **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So to handle a big project without any problem, the company has to go for a software engineering method.
5. **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
6. **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So Software becomes more effective in the act with the help of software engineering.

Software Processes

The term **software** specifies to the set of computer programs, procedures and associated documents (Flowcharts, manuals, etc.) that describe the program and how they are to be used.
A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. These activities are:
1. **Software specifications:** The functionality of the software and constraints on its operation must be defined.
2. **Software development:** The software to meet the requirement must be produced.
3. **Software validation:** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution:** The software must evolve to meet changing client needs.

The Software Process Model

A software process model is a specified definition of a software process, which is presented from a particular perspective. Models, by their nature, are a simplification, so a software process model is an abstraction of the actual process, which is being described. Process models may contain activities, which are part of the software process, software product, and the roles of people involved in software engineering. Some examples of the types of software process models that may be produced are:

1. **A workflow model:** This shows the series of activities in the process along with their inputs, outputs and dependencies. The activities in this model perform human actions.
2. **2. A dataflow or activity model:** This represents the process as a set of activities, each of which carries out some data transformations. It shows how the input to the process, such as a specification is converted to an output such as a design. The activities here may be at a lower level than activities in a workflow model. They may perform transformations carried out by people or by computers.
3. **3. A role/action model:** This means the roles of the people involved in the software process and the activities for which they are responsible.

There are several various general models or paradigms of software development:

1. **The waterfall approach:** This takes the above activities and produces 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 onto the following stage.
2. **Evolutionary development:** This method interleaves the activities of specification, development, and validation. An initial system is rapidly developed from a very abstract specification.
3. **Formal transformation:** This method is based on producing a formal mathematical system specification and transforming this specification, using mathematical methods to a program. These transformations are 'correctness preserving.' This means that you can be sure that the developed programs meet its specification.
4. **System assembly from reusable components:** This method assumes the parts of the system already exist. The system development process target on integrating these parts rather than developing them from scratch.

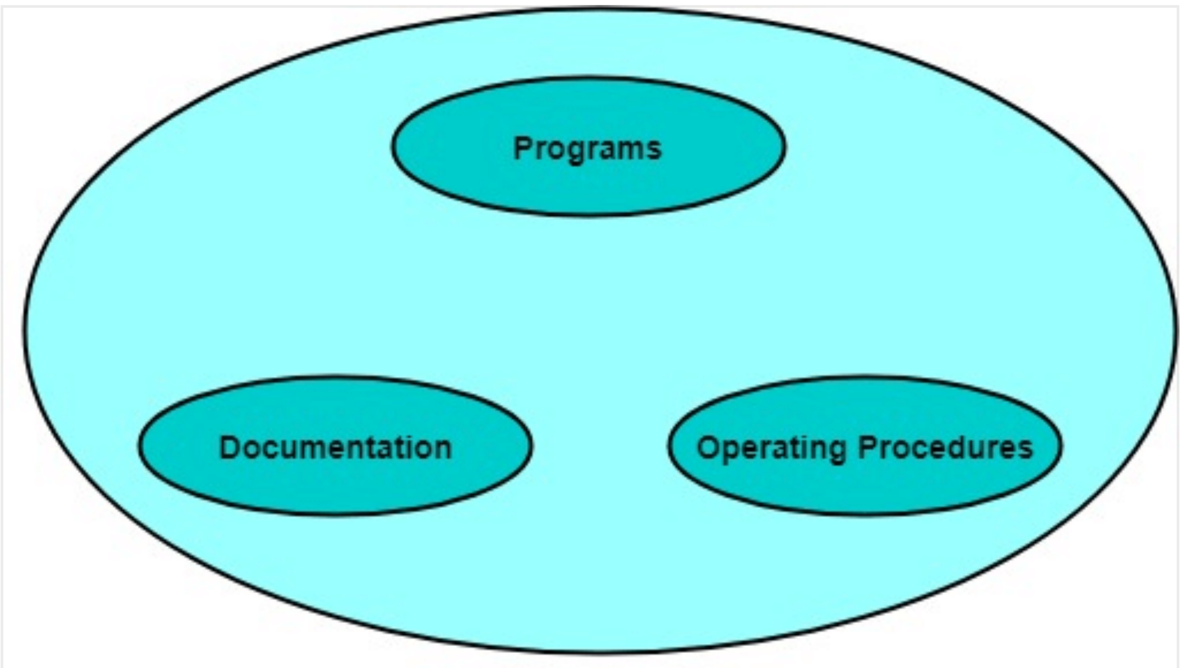
Software Crisis

1. **Size:** Software is becoming more expensive and more complex with the growing complexity and expectation out of software. For example, the code in the consumer product is doubling every couple of years.
2. **Quality:** Many software products have poor quality, i.e., the software products defects after putting into use due to ineffective testing technique. For example, Software testing typically finds 25 errors per 1000 lines of code.

3. **Cost:** Software development is costly i.e. in terms of time taken to develop and the money involved. For example, Development of the FAA's Advanced Automation System cost over $700 per lines of code.
4. **Delayed Delivery:** Serious schedule overruns are common. Very often the software takes longer than the estimated time to develop, which in turn leads to cost shooting up. For example, one in four large-scale development projects is never completed.

Program vs. Software

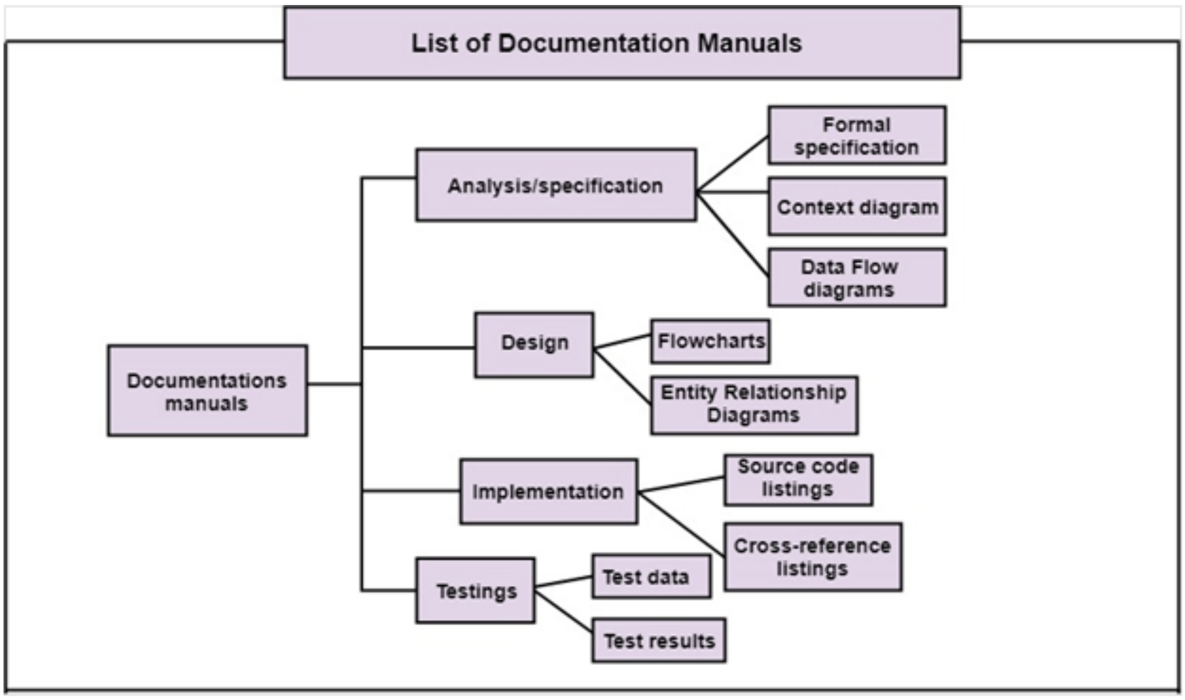
Software is more than programs. Any program is a subset of software, and it becomes software only if documentation & operating procedures manuals are prepared.
There are three components of the software as shown in fig:



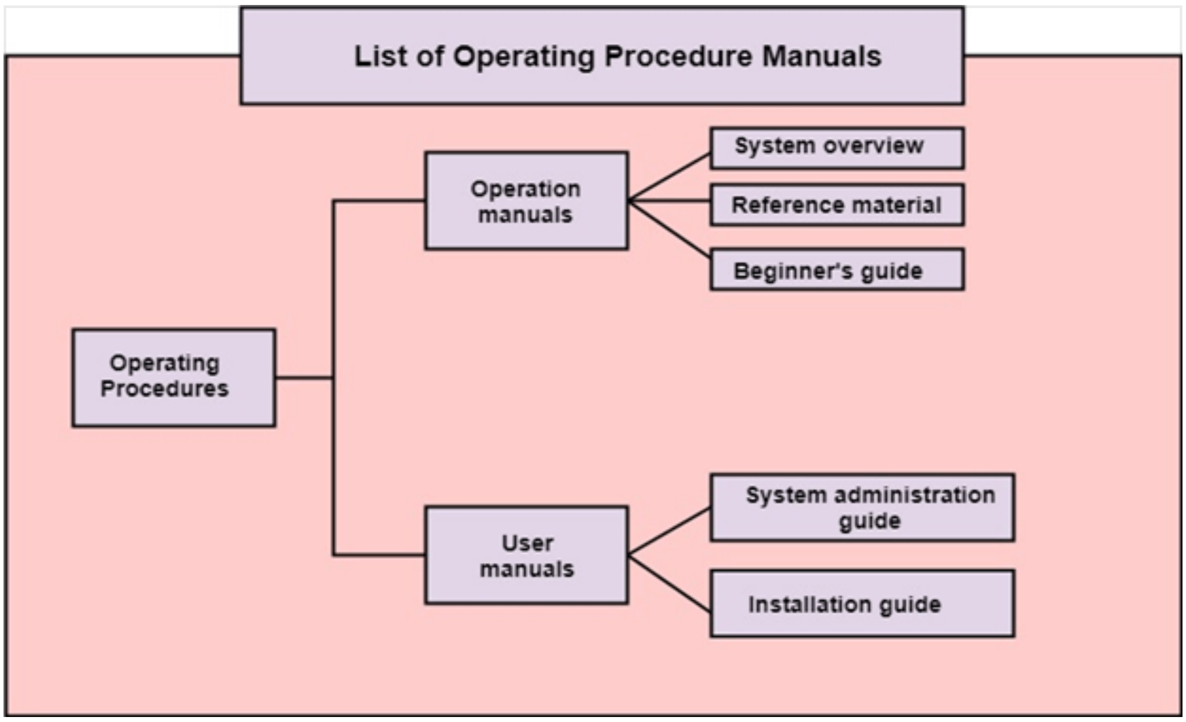
Software= Program + Documentation + Operating Procedures

Fig:Components of Software

1. Program: Program is a combination of source code & object code.
2. Documentation: Documentation consists of different types of manuals. Examples of documentation manuals are: Data Flow Diagram, Flow Charts, ER diagrams, etc.



3. Operating Procedures: Operating Procedures consist of instructions to set up and use the software system and instructions on how react to the system failure. Example of operating system procedures manuals is: installation guide, Beginner's guide, reference guide, system administration guide, etc.



Software Development Life Cycle (SDLC)

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

Need of SDLC

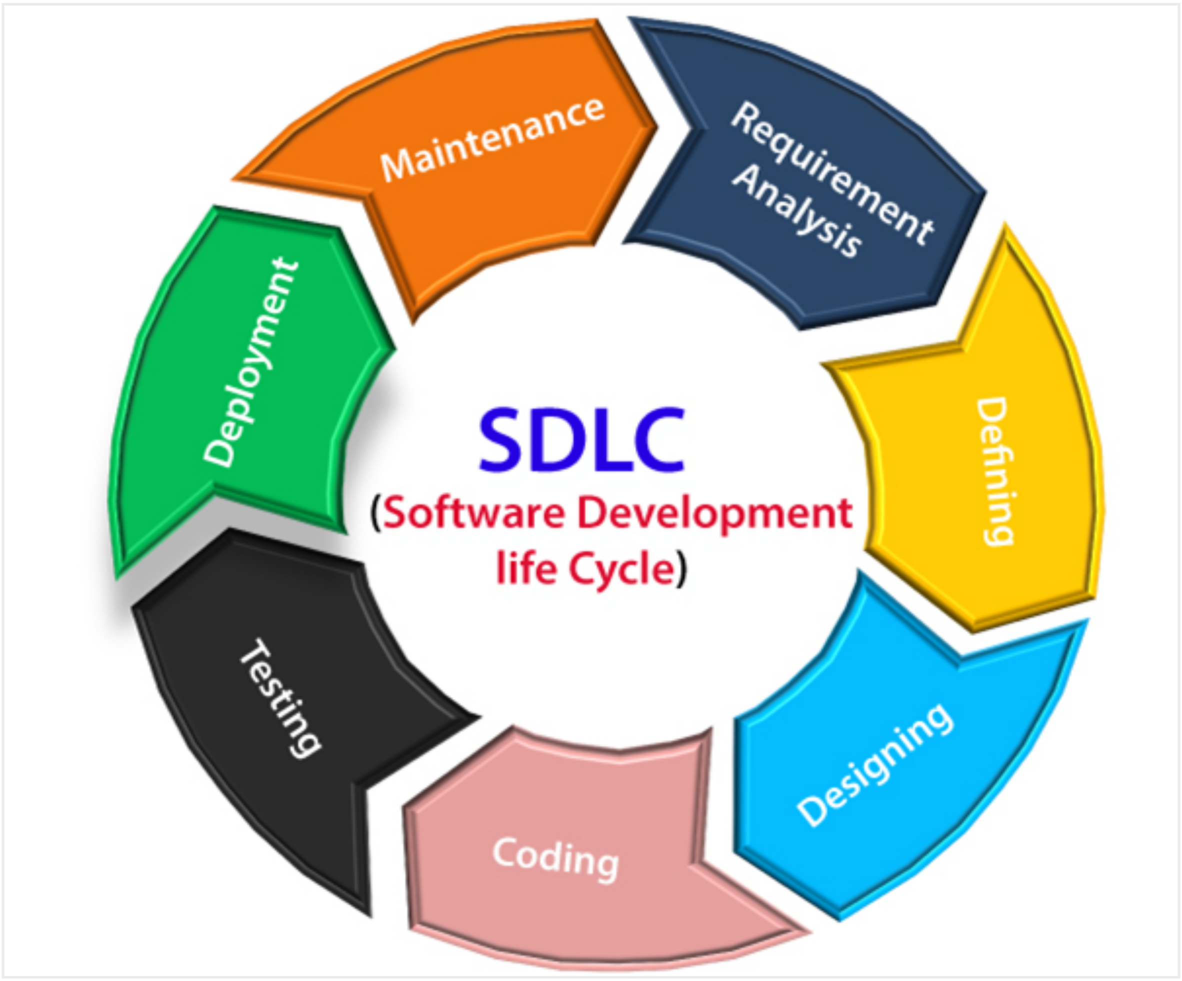
The development team must determine a suitable life cycle model for a particular plan and then observe to it.

Without using an exact life cycle model, the development of a software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team representative about when and what to do. Otherwise, it would point to chaos and project failure. This problem can be defined by using an example. Suppose a software development issue is divided into various parts and the parts are assigned to the team members. From then on, suppose the team representative is allowed the freedom to develop the roles assigned to them in whatever way they like. It is possible that one representative might start writing the code for his part, another might choose to prepare the test documents first, and some other engineer might begin with the design phase of the roles assigned to him. This would be one of the perfect methods for project failure.

A software life cycle model describes entry and exit criteria for each phase. A phase can begin only if its stage-entry criteria have been fulfilled. So without a software life cycle model, the entry and exit criteria for a stage cannot be recognized. Without software life cycle models, it becomes tough for software project managers to monitor the progress of the project.

SDLC Cycle

SDLC Cycle represents the process of developing software. SDLC framework includes the following steps:



The stages of SDLC are as follows:

Stage1: Planning and requirement analysis

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

Stage2: Defining Requirements

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

Stage3: Designing the Software

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

Stage4: Developing the project

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

Stage5: Testing

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.

During this stage, unit testing, integration testing, system testing, acceptance testing are done.

Stage6: Deployment

Once the software is certified, and no bugs or errors are stated, then it is deployed.

Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

Stage7: Maintenance

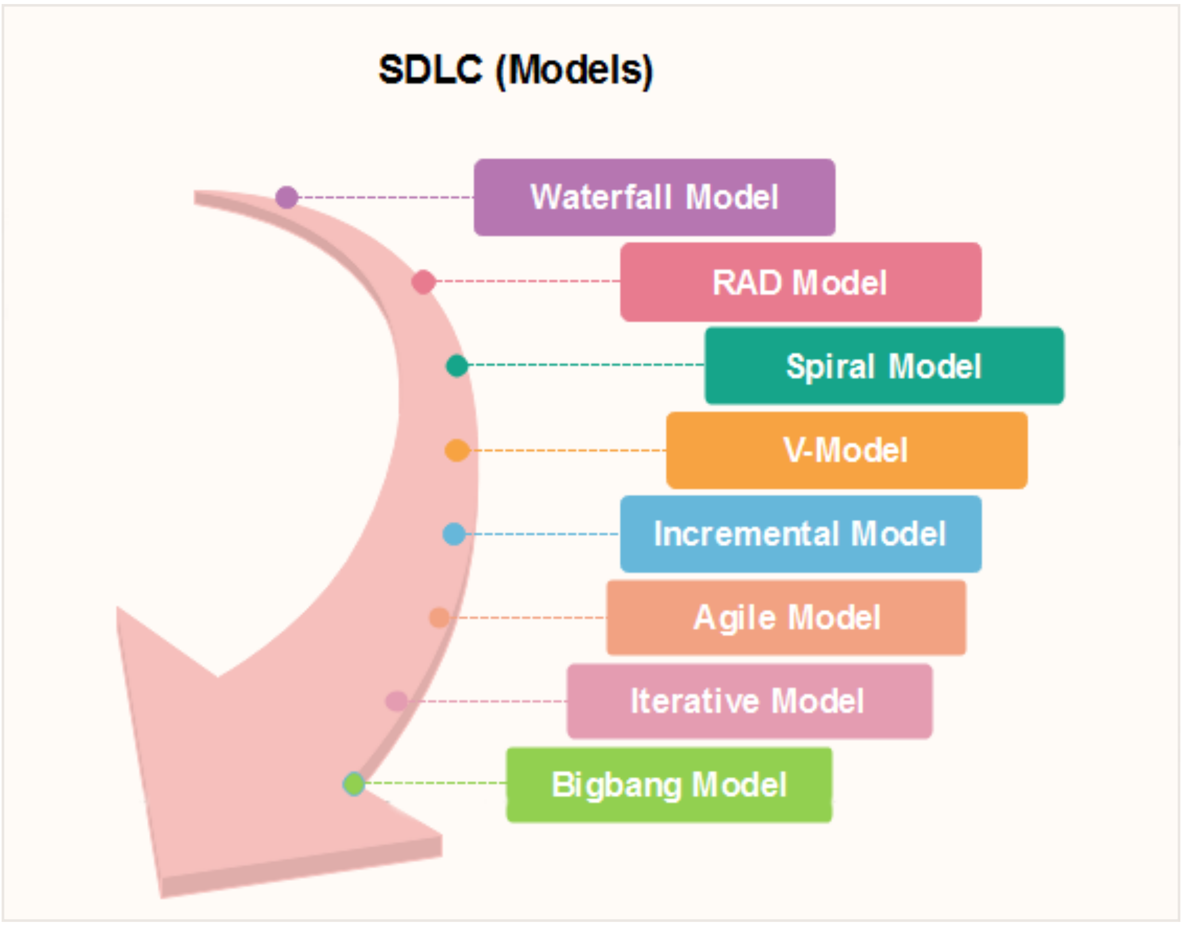
Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

This procedure where the care is taken for the developed product is known as maintenance.

SDLC Models

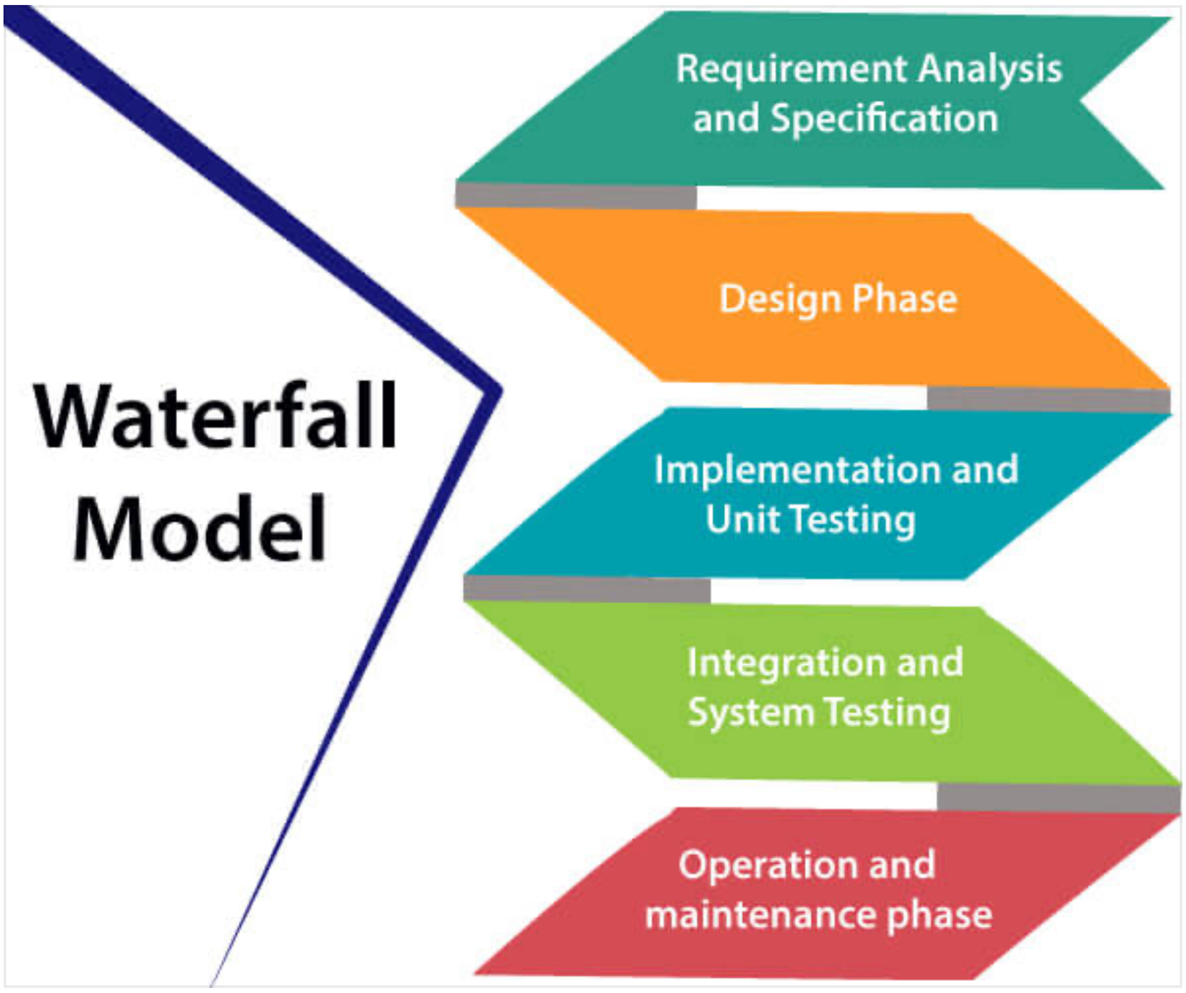
Software Development life cycle (SDLC) is a spiritual model used in

project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.
There are different software development life cycle models specify and design, which are followed during the software development phase. These models are also called "**Software Development Process Models**." Each process model follows a series of phase unique to its type to ensure success in the step of software development.
Here, are some important phases of SDLC life cycle:



Waterfall model
Winston Royce introduced the Waterfall Model in 1970.This model has five phases: Requirements analysis and specification, design, implementation, and unit testing, integration and system testing, and operation and maintenance. The steps always follow in this order and do not overlap. The developer must complete every phase before the next phase begins. This model is named "**Waterfall Model**", because its diagrammatic representation resembles a cascade of waterfalls.
1. Requirements analysis and specification phase: The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions,

performance, and interfacing requirement of the software. It describes the "what" of the system to be produced and not "how."In this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.


Waterfall Model
Requirement Analysis and Specification
Design Phase
Implementation and Unit Testing
Integration and System Testing
Operation and maintenance phase

2. Design Phase: This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

3. Implementation and unit testing: During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.

During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

4. Integration and System Testing: This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

5. Operation and maintenance phase: Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

When to use SDLC Waterfall Model?

Some Circumstances where the use of the Waterfall model is most suited are:

o When the requirements are constant and not changed regularly.

o A project is short

o The situation is calm

o Where the tools and technology used is consistent and is not changing

o When resources are well prepared and are available to use.

Advantages of Waterfall model

o This model is simple to implement also the number of resources that are required for it is minimal.

o The requirements are simple and explicitly declared; they remain unchanged during the entire project development.

o The start and end points for each phase is fixed, which makes it easy to cover progress.

o The release date for the complete product, as well as its final cost, can be determined before development.

o It gives easy to control and clarity for the customer due to a strict reporting system.

Disadvantages of Waterfall model

o In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.

o This model cannot accept the changes in requirements during development.

o It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.

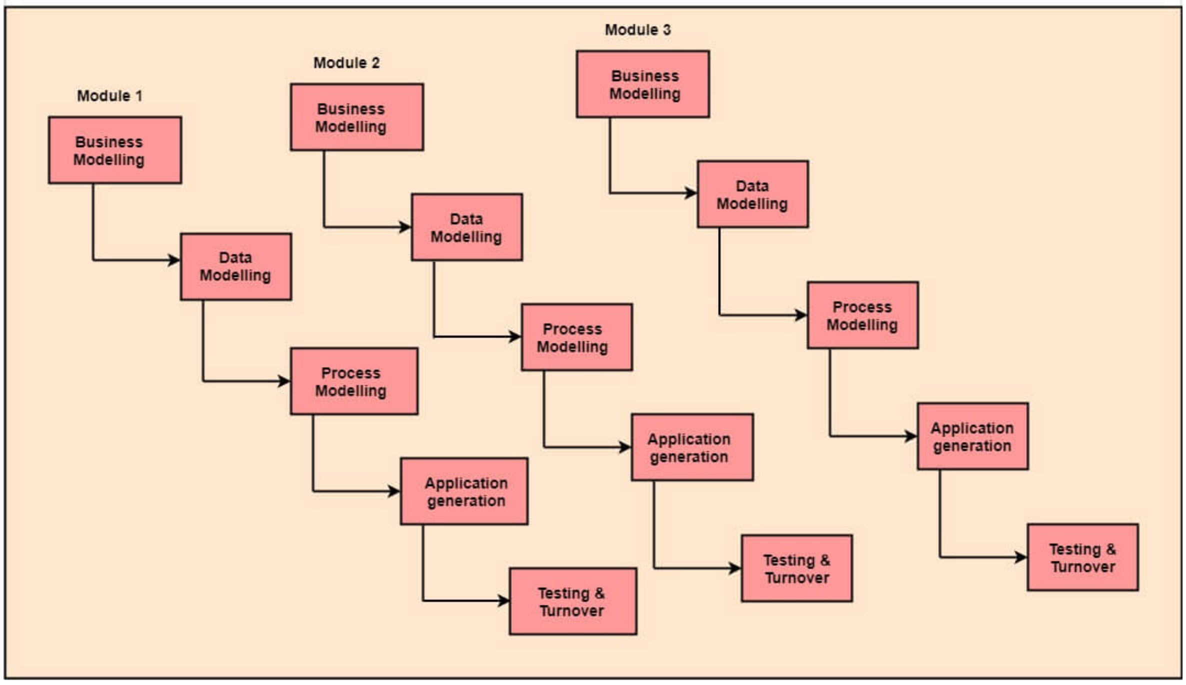
o Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

RAD (Rapid Application Development) Model

RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based

construction approach. If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.
RAD (Rapid Application Development) is a concept that products can be developed faster and of higher quality through:
o Gathering requirements using workshops or focus groups
o Prototyping and early, reiterative user testing of designs
o The re-use of software components
o A rigidly paced schedule that refers design improvements to the next product version
o Less formality in reviews and other team communication

Fig: RAD Model



The various phases of RAD are as follows:
1.Business Modelling: The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.
2. Data Modelling: The data collected from business modeling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.
3. Process Modelling: The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.
4. Application Generation: Automated tools are used to facilitate

construction of the software; even they use the 4th GL techniques.

5. Testing & Turnover: Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

When to use RAD Model?

o When the system should need to create the project that modularizes in a short span time (2-3 months).

o When the requirements are well-known.

o When the technical risk is limited.

o When there's a necessity to make a system, which modularized in 2-3 months of period.

o It should be used only if the budget allows the use of automatic code generating tools.

Advantage of RAD Model

o This model is flexible for change.

o In this model, changes are adoptable.

o Each phase in RAD brings highest priority functionality to the customer.

o It reduced development time.

o It increases the reusability of features.

Disadvantage of RAD Model

o It required highly skilled designers.

o All application is not compatible with RAD.

o For smaller projects, we cannot use the RAD model.

o On the high technical risk, it's not suitable.

o Required user involvement.

Spiral Model

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced.

The Spiral Model is shown in fig:

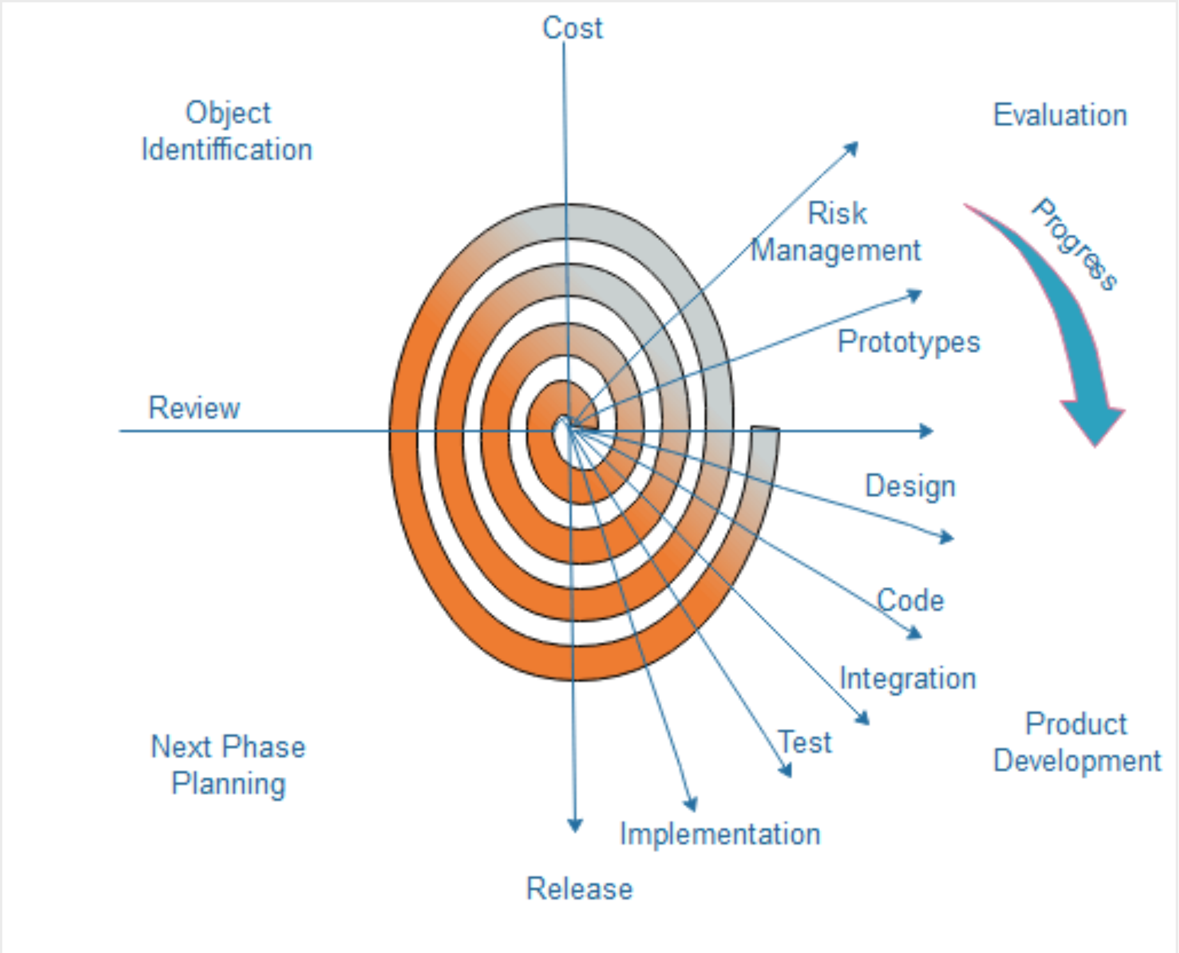


Fig. Spiral Model

Each cycle in the spiral is divided into four parts:

Objective setting: Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.

Risk Assessment and reduction: The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.

Development and validation: The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.

Planning: Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

The development phase depends on the remaining risks. For example, if performance or user-interface risks are treated more essential than the program development risks, the next phase may be an evolutionary development that includes developing a more

detailed prototype for solving the risks.
The **risk-driven** feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.
When to use Spiral Model?
o When deliverance is required to be frequent.
o When the project is large
o When requirements are unclear and complex
o When changes may require at any time
o Large and high budget projects
Advantages of the Spiral Model
There are so many benefits of using the spiral model. These are as follows:
o The amount of risk should be reduced in this model.
o This model is best for large and critical projects.
o It provides the developer with solid approval and documentation control.
o The software production time is less in this spiral model.
Disadvantages of Spiral Model
It also has some disadvantages to using the spiral model. These are as follows.
o We need a lot of financial support to develop a project.
o We should not use this model for a very small project.

Incremental Model
Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

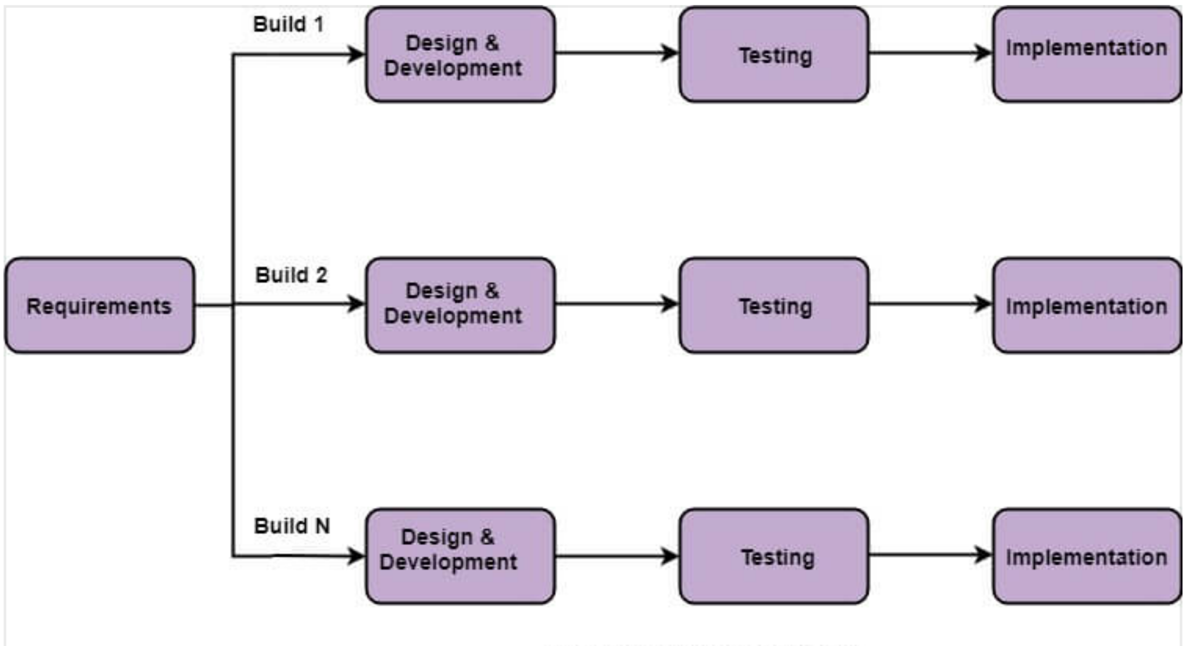


Fig: Incremental Model

The various phases of incremental model are as follows:

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

ADVERTISEMENT

o When the requirements are superior.

o A project has a lengthy development schedule.

o When Software team are not very well skilled or trained.

o When the customer demands a quick release of the product.

o You can develop prioritized requirements first.

Advantage of Incremental Model

o Errors are easy to be recognized.
o Easier to test and debug
o More flexible.
o Simple to manage risk because it handled during its iteration.
o The Client gets important functionality early.

Disadvantage of Incremental Model

o Need for good planning
o Total Cost is high.
o Well defined module interfaces are needed.

Iterative Model

In this Model, you can start with some of the software specifications and develop the first version of the software. After the first version if there is a need to change the software, then a new version of the software is created with a new iteration. Every release of the Iterative Model finishes in an exact and fixed period that is called iteration.
The Iterative Model allows the accessing earlier phases, in which the variations made respectively. The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process.

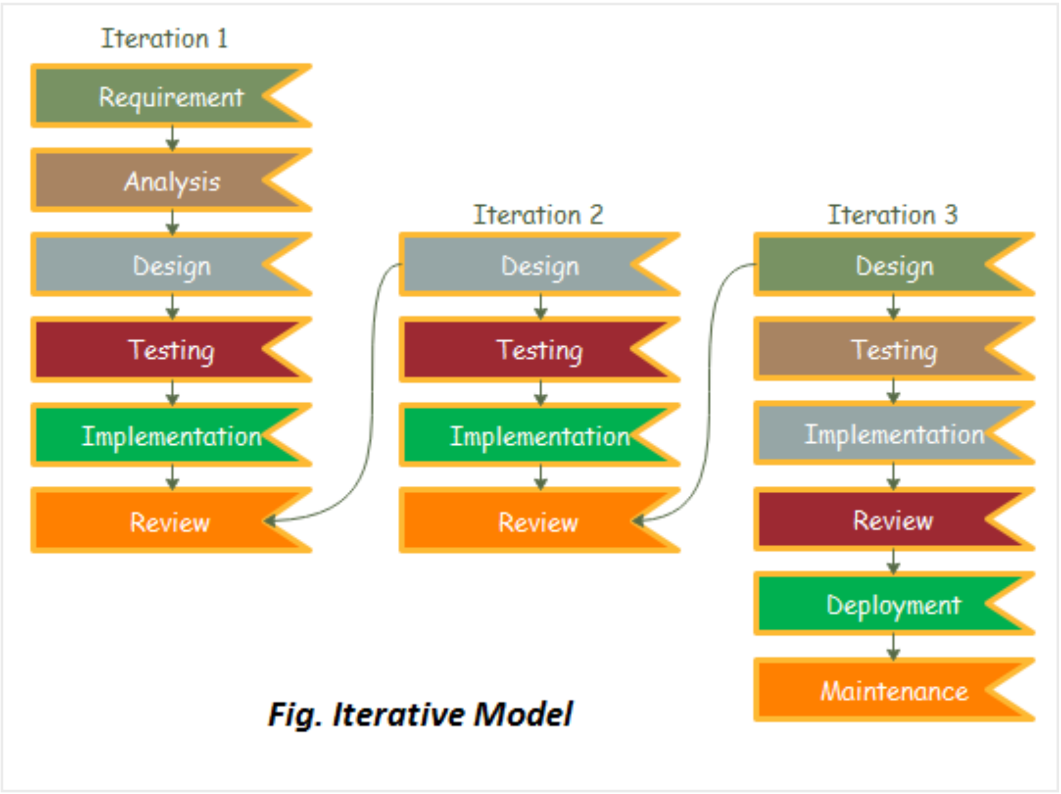


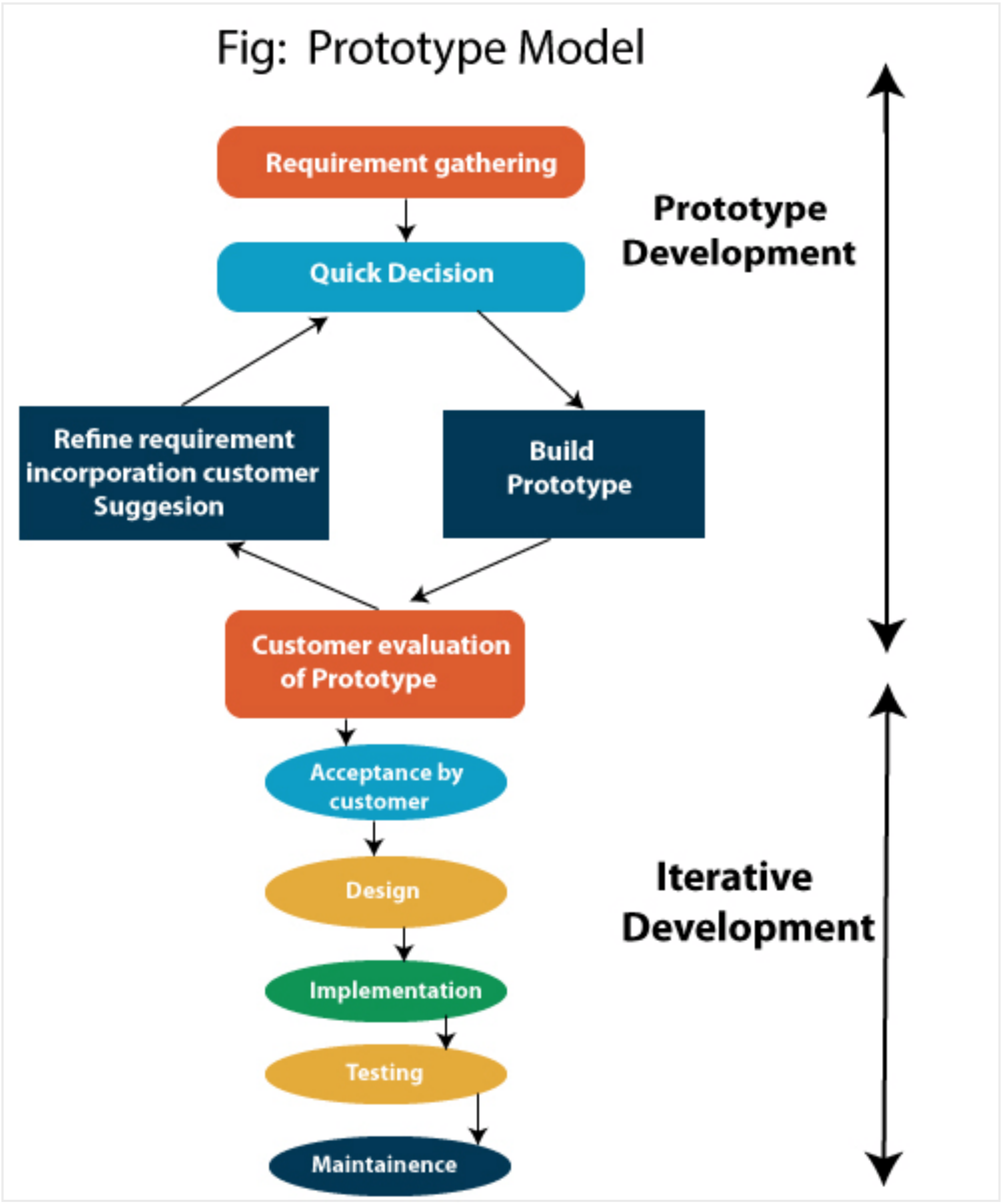
Fig. Iterative Model

The various phases of Iterative model are as follows:

1. Requirement gathering & analysis: In this phase, requirements are gathered from customers and check by an analyst whether requirements will fulfil or not. Analyst checks that need will achieve

within budget or not. After all of this, the software team skips to the next phase.
2. Design: In the design phase, team design the software by the different diagrams like Data Flow diagram, activity diagram, class diagram, state transition diagram, etc.
3. Implementation: In the implementation, requirements are written in the coding language and transformed into computer programmes which are called Software.
4. Testing: After completing the coding phase, software testing starts using different test methods. There are many test methods, but the most common are white box, black box, and grey box test methods.
5. Deployment: After completing all the phases, software is deployed to its work environment.
6. Review: In this phase, after the product deployment, review phase is performed to check the behaviour and validity of the developed product. And if there are any error found then the process starts again from the requirement gathering.
7. Maintenance: In the maintenance phase, after deployment of the software in the working environment there may be some bugs, some errors or new updates are required. Maintenance involves debugging and new addition options.
When to use the Iterative Model?
1. When requirements are defined clearly and easy to understand.
2. When the software application is large.
3. When there is a requirement of changes in future.
Advantage(Pros) of Iterative Model:
1. Testing and debugging during smaller iteration is easy.
2. A Parallel development can plan.
3. It is easily acceptable to ever-changing needs of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.
Disadvantage(Cons) of Iterative Model:
1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.
Prototype Model
The prototype model requires that before carrying out the

development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.



Steps of Prototype Model

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

Advantage of Prototype Model

1. Reduce the risk of incorrect user requirement
2. Good where requirement are changing/uncommitted
3. Regular visible process aids management
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.

Disadvantage of Prototype Model

1. An unstable/badly implemented prototype often becomes the final product.
2. Require extensive customer collaboration
o Costs customer money
o Needs committed customer
o Difficult to finish if customer withdraw
o May be too customer specific, no broad market
3. Difficult to know how long the project will last.
4. Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5. Prototyping tools are expensive.
6. Special tools & techniques are required to build a prototype.
7. It is a time-consuming process.

Evolutionary Process Model

Evolutionary process model resembles the iterative enhancement model. The same phases are defined for the waterfall model occurs here in a cyclical fashion. This model differs from the iterative enhancement model in the sense that this does not require a useful product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

For example, in a simple database application, one cycle might implement the graphical user Interface (GUI), another file manipulation, another queries and another updates. All four cycles must complete before there is a working product available. GUI allows the users to interact with the system, file manipulation allow the data to be saved and retrieved, queries allow user to get out of the system, and updates allows users to put data into the system.

Benefits of Evolutionary Process Model

Use of EVO brings a significant reduction in risk for software projects.

EVO can reduce costs by providing a structured, disciplined avenue for experimentation.

EVO allows the marketing department access to early deliveries, facilitating the development of documentation and demonstration.

Better fit the product to user needs and market requirements.

Manage project risk with the definition of early cycle content.

Uncover key issues early and focus attention appropriately.

Increase the opportunity to hit market windows.

Accelerate sales cycles with early customer exposure.

Increase management visibility of project progress.

Increase product team productivity and motivations.

Characteristics of the Evolutionary Model

There are so many characteristics of using the evolutionary model in our project. These characteristics are as follows.

o We can develop the evolutionary model with the help of an iterative waterfall model of development.

o There are three types of evolutionary models. These are the Iterative model, Incremental model and Spiral model.

o Many primary needs and architectural planning must be done

o to implement the evolutionary model.

o When the new product version is released, it includes the new functionality and some changes in the existing product, which are also released with the latest version.

o This model also permits the developer to change the requirement, and the developer can divide the process into different manageable work modules.

o The development team also have to respond to customer feedback throughout the development process by frequently altering the product, strategy, or process.

Advantages of the Evolutionary Model

There are so many benefits of using the evolutionary model. These benefits are as follows.

o In the evolutionary model, the user can test partially developed software before the release of the final product in the market.

o In this model, we can test the core module, which reduces the chances of errors in the final product.

o We can use this module only for large products.

o We can use this model to develop a software product with some unique features. We can modify these specific features based on customer feedback and many factors throughout development.

o With the help of this evolutionary model, we can get to know the client's requirements during the delivery of different versions of the software.
o After completion of each cycle, the user can access the product.
o Using an evolutionary model removes the requirement to allocate significant resources simultaneously for system development.

Disadvantages of the Evolutionary Model

There are also many disadvantages to using this model. These are as follows.
o It is difficult to divide the problem into functionality units that the user accepts. After receiving the user, this problem should be incrementally implemented, and after that, it should be delivered.
o There will be a chance of being late in the delivery of the final product. There will be a chance that the market can go down due to different customer changes during development.
o In this module, the chances of risk factors are high. There is always a need to report customers continuously.

Comparison of the Evolutionary Model with Other Models

1. Evolutionary Model vs Incremental Model

Evolutionary	**Incremental**
The requirement of this evolutionary model is not a requirement. The development can be changed according to the development process.	The requirements for this incremental model are precise and based on the development team.
In this evolutionary model, the initial step is understanding the customer's requirements. Also, it involves the development of the core modules and functionality. After completing all the phases, they have to deliver the product to the customer for feedback.	In this incremental model, each module is defined with multiple iterations. We can also add new functionality during the development of the product. After completing all the steps, we have to release the deliverable product.
The activities of the complete cycle are repeated for each new product version.	Each module is developed, tested, and released at various intervals.

In this model, the development time is unknown for the developer. Also, the developer can not track the progress of the software product.	In this model, the development time is known to the developer. Also, the developer can track the progress of the software product.

2. Evolutionary Model vs Iterative Model

Evolutionary Model	**Iterative Model**
In this model, different modules are released in incremental order before releasing the final product.	It follows the process of the sequential development process. After completion of the development of the software product, it releases the final product.
There will be a chance of delivery of actual product be late.	In this model, the product will be delivered within the time.
In this module, the integration of the module will be complex.	This model makes it much easier to understand and implement the product.
We can use this module only for large products.	It is the most commonly used module in the software industry.

3. Evolutionary Model vs Classical Waterfall Model

Evolutionary Model	**Classical Waterfall Model**
In this model, we can deal with the different versions of the software.	In this model, we can deal with things according to the things mentioned in the testing manual.
In this model, integration is going to be complicated.	In this model, it is going to be simple to use, understand and implement.
This model accepts customer feedback during the development of the project.	In this model, there is no need for customer feedback.
In this model, we can detect the error in the core module of the project.	In this phase, we can detect the error in every development step.

If we need to add any new functionality, we have to release the latest product, and the existing functionalities may be updated.	After the completion of the development of the final product, we can not add any update to it.

4. Evolutionary Model vs Spiral Model

Evolutionary Model	**Spiral Model**
In this model, we can develop the software incrementally with the help of different modules.	In this model, all the modules are divided into many loops, further divided into four quadrants.
We can use this module only for large products.	We can use this model to develop technically difficult software products vulnerable to different threats.
In this model, Every version can fully function the mentioned functionalities.	In this module, each model contains a set of activities that the software performs.

Advantages of Software Engineering
There are far more benefits of software engineering than drawbacks, and a large number of individuals work in the industry. We are aware of how crucial software engineering is to the creation and upkeep of software systems. **The benefits of software engineering are as follows:**
1. Huge Requirement
The competition for jobs continually has a strong demand for software developers. The demand for qualified software engineers is growing as organizations increasingly depend on software solutions. For software developers, this need leads to various work options and potential for professional advancement.
2. Adjustable time-schedule
Software engineering frequently provides freedom to plan. Many software developers can choose to operate online or on their schedule. This adaptability enables professionals to regulate their private and work lives better, improving work-life balance.
3. Excellent Programmes
Software engineering approaches and practices strongly emphasize the creation of software programs of the highest caliber. Software engineers may produce dependable and durable software solutions

that fulfill user expectations by conducting an organized evaluation of requirements, rigorous testing, and code reviews. Better user and consumer experiences result from this focus on quality.

4. Huge Earning

Relative to several other careers, software engineering is regarded as having the potential for higher income. The higher revenue potential is frequently a result of the need for qualified software engineers and their specialized knowledge and expertise. For those looking for a job with competitive pay, software engineering is an appealing option.

5. Increased Employment possibilities

Nowadays, various sectors, including banking, healthcare, e-commerce, and entertainment, require experts in software engineering. The job prospects of software engineers are improved by the wide range of options that guarantee they have a diversified choice of businesses and organizations to work with.

6. The All-Directional Permanent Learning Curve

New technologies, frameworks, and approaches are continually being introduced in the field of software engineering. For software developers, this creates an engaging and ongoing learning curve. Whether it's a programming language, a development tool, or a new tech trend, there's always something new to learn and explore. Software developers may stay on the cutting edge of technological breakthroughs because of their ongoing learning and intellectual engagement.

7. Business possibilities

In the current digital era, software engineering talents are in high demand. Software developers now have a wide range of business options. They might explore entrepreneurship by establishing their own software development businesses or consulting businesses.

They can also work with established companies and startups to create cutting-edge software solutions. Professionals can explore new business opportunities and have a big effect across multiple industries thanks to the demand for software engineering talent.

These are the advantages of Software engineering. Now, let us discuss the disadvantages of software engineering.

Disadvantages of Software Engineering

There are many disadvantages that can be listed in software engineering. Every field has benefits and drawbacks. We need to survive all those things to get a better life. **The following are the disadvantages of Software Engineering:**

1. Health Problems Because of Longer Working Periods:

Long periods of time spent in front of a computer by software engineers can cause a number of health concerns, including back discomfort, eye strain, and disorders associated with a sedentary lifestyle. It is crucial that software developers put their health first, make time for regular exercise, and follow ergonomic guidelines.

2. Project timelines that are difficult to meet:

Software engineering projects frequently have rushed completion dates and stressful work conditions. It can be challenging to meet project deadlines and deliver on schedule, which can result in stress and burnout. These difficulties can be lessened with good time management, thorough preparation, and reasonable expectations.

3. Periodical updates in Technology:

Technology is always changing, and the software engineering industry is no exception. It may be difficult to stay current with the newest programming languages, frameworks, and tools. To stay competitive, software engineers must regularly upgrade their knowledge and follow market trends.

4. Security Concerns:

As our reliance on software systems grows, security flaws and online dangers have taken on greater importance. Implementing strong security measures and having a thorough grasp of potential threats is essential for creating secure software. Inadequate security measures can lead to data breaches and jeopardize user privacy.

5. Price issues:

Software system development and upkeep can be costly. Hardware, software licenses, development tools, and ongoing maintenance are frequently expensive components of software engineering projects. Budget restraints and cost overruns can be problematic for businesses and have an impact on the success of projects.

6. Restricted Monitoring:

The project scope, decision-making process, and project management elements of software projects may be subject to restricted software engineer control. They could have to operate within the limitations imposed by stakeholders or project managers, which may limit their creativity and liberty.

7. Not enough Assistance:

Software engineers may experience an insufficient amount of mentoring, resources, or support in some organizations. This may impede their capacity to advance professionally and efficiently handle difficult problems. To get over these constraints, software developers must actively seek out learning and growth opportunities.

Need for Software Engineering

Software Engineering is a branch of Engineering where you study how to develop Software. It is a branch of Computer science where you design, develop, test and maintain the Software according to the user's requirement to solve real-world problems. Software Engineering Consists of two words, Software which means applications that have been designed using a set of rules and regulations, and Engineering means to invent, design, build, and maintain the application.

Software Engineering has become the most important choice for many, and the reason is that every organization needs a software engineer. In today's World, Software engineering has not only been serving the need of IT Companies but there is a need for software engineering to solve daily life problems.

Software Engineering has many needs in our era. Following are some points that tell us about the need for Software Engineering:

o **Building Social Media Platform:** Software Engineering is needed in building social media platforms. It can be used to design user profiles, feed the news, message, and many more personalization.

o **Building E-Learning Platform:** Software Engineering is needed in building educational Software and creating E-Learning Platform. It is used to create interactive learning modules and interactive courses.

o **Transportation system:** Software Engineering can be used to build Software that helps track the route. It includes real-time data management and GPS and Mapping Tools.

o **Healthcare Software:** Software engineering is needed in developing Software for healthcare systems. It is needed in preparing electronic health records (EHR) systems, telemedicine applications, medical imaging software, and clinical decision support systems. Software engineering ensures that the accuracy provided by the Software is 100 percent correct and there are no loopholes. It is also needed to maintain the privacy and security of patient data.

o **Financial Software:** Software Engineering is needed in preparing Software for Banking and other Financial Organisations. Software Engineering helps maintain security and data accuracy so that the user's financial data may not leak.

o **Mobile Application Development:** Software engineering is needed to develop mobile application platforms like iOS and Android. It is used to design user interfaces. It is used to ensure compatibility among different devices.

o **Video Game Development:** Software Engineering is needed in developing Video Games. It is helpful in developing the game's design, setting the game's graphics, allowing multiplayer networking,

etc.
o **Research:** It helps to visualize data efficiency and process extensive data.
o **Artificial Intelligence and Machine Learning:** It helps in designing and implementing algorithms, model training, and deploy machine learning models to solve complex problems and enable intelligent and faster decision-making.
o **Application Security:** It helps maintain the security of the Software. It allows application security in many ways. Software engineering techniques allow the software developer to encrypt the data so that no unauthorized person may not be able to look into the data, or if he can get a hand on the data, he would not be able to decode that data. In this way, the user's data would be safe. It allows the features like authentication, where the user can enter his details. After entering his details, software engineering techniques verify the data, and if the entered credentials match that saved earlier, then the user is allowed to enter his account. This saves the user from the third party to access their account. Thus, software engineering is beneficial in ensuring the confidentiality of user accounts.
o **Fixing Bugs and Errors:** Software Engineering is beneficial in updating the existing Software. It is also used in adding any new features to the application. It also upgrades the application when needed or when the application is outdated.
Software Myths in Software Engineering
Software Myths are beliefs that do not have any pure evidence. Software myths may lead to many misunderstandings, unrealistic expectations, and poor decision-making in software development projects. Some common software myths include:

o **The Myth of Perfect Software:** Assuming that it's possible to create bug-free software. In Reality, software is inherently complex, and it's challenging to eliminate all defects.
o **The Myth of Short Development Times:** Assuming that software can be developed quickly without proper planning, design, and testing. In Reality, rushing the development process can lead to better-quality software and missed deadlines.
o **The Myth of Linear Progression:** The Development of software is in a linear, predictable manner. In Reality, development is often iterative and can involve unexpected setbacks and changes.
o **The Myth of No Maintenance:** It is thought that software development is complete once the initial version is released. But in reality, the software requires maintenance and updates to remain

functional and secure.

o **The Myth of User-Developer Mind Reading:** It is assumed that developers can only understand user needs with clear and ongoing communication with users. But in reality, user feedback and collaboration are essential for correct software development.

o **The Myth of Cost Predictability:** It is thought that the cost of the software can be easily predicted, but in reality, many factors can influence project costs, and estimates are often subject to change. There are many hidden costs available.

o **The Myth of Endless Features:** It is believed that adding more features to software will make it better. But in reality, adding more features to the software can make it complex and harder to use and maintain. It may often lead to a worse user experience.

o **The Myth of No Testing Needed:** It is assumed that there is no need to test the software if the coder is skilled or the code looks good. But in reality, thorough testing is essential to catch hidden defects and ensure software reliability.

o **The Myth of One-Size-Fits-All Methodologies:** Thinking that a single software development methodology is suitable for all projects. But in reality, different methodologies should be used on the specific project.

o **The Myth of "We'll Fix It Later":** It is assumed that a bug can be fixed at a later stage. But in reality, as the code gets longer and bigger, it takes a lot of work to find and fix the bug. These issues can lead to increased costs and project delays.

o **The Myth of All Developers Are Interchangeable:** It is believed that any developer can replace another without any impact. But in reality, each developer has unique skills and knowledge that can significantly affect the project. Each one has a different method to code, find, and fix the bugs.

o **The Myth of No User Training Required:** It is assumed that users will understand and use new software without any training. But in reality, users need training and documentation to use the new software because the different methods used by different developers can be unique.

o **More Developers Equal Faster Development:** It is believed that if there are large no of coders, then the software development would take less time, and the quality of the software would be of high quality. But in reality, larger teams can lead to communication overhead and may only sometimes result in faster development.

o **Perfect Software Is Possible:** The Idea of creating completely bug-free software is a myth. In Reality, software development is

complex, and it's challenging to eliminate all defects.
o **Zero-Risk Software:** It is assumed that it's possible to develop software with absolutely no risks. But in reality, all software projects involve some level of risk, and risk management is a critical part of software development.
Disadvantages of Software Myths
Software myths in software engineering can have several significant disadvantages and negative consequences, as they can lead to unrealistic expectations, poor decision-making, and a lack of alignment between stakeholders. Here are some of the disadvantages of software myths in software engineering:
o **Unrealistic Expectations:** Software myths can create disappointment and frustration among the stakeholders and developers. Sometimes, the fake myth may lead to the no use of the software when it is completely safe.
o **Project Delays:** Software myths will lead to more delays in the completion of the projects and also increase the completion time of the projects.
o **Poor Quality Software:** Myths such as "we can fix it later" or "we don't need extensive testing" can lead to poor software quality. Neglecting testing and quality assurance can result in buggy and unreliable software.
o **Scope Creep:** Myths like "fixed requirements" can lead to scope creep as stakeholders may change their requirements or expectations throughout the project. This can result in a never-ending development cycle.
o **Ineffective Communication:** Believing in myths can affect good communication within development teams and between teams and clients. Clear and open communication is crucial for project success, and myths can lead to misunderstandings and misalignment.
o **Wasted Resources:** The Idea of getting "the perfect software" can result in the allocation of unnecessary resources, both in terms of time and money, which could be better spent elsewhere.
o **Customer Dissatisfaction:** Unrealistic promises made based on myths can lead to customer dissatisfaction. When software doesn't meet exaggerated expectations, clients may be disappointed and dissatisfied.
o **Reduced Productivity:** Myths can lead to reduced productivity, as team members may spend time on unnecessary tasks or follow counterproductive processes based on these myths.
o **Increased Risk of Project Failure:** The reliance on myths can significantly increase the risk of project failure. Failure to address

these myths can lead to project cancellations, loss of investments, and negative impacts on an organization's reputation.

o **Decreased Competitiveness:** Belief in myths can make an organization less competitive in the market. It can hinder an organization's ability to innovate and adapt.

Types of Software Myths

Software myths can take various forms and cover a wide range of misconceptions and misunderstandings in software engineering. Here are some common types of software myths:

Productivity Myths:

More Developers can increase productivity and will give a better software product.

Quality Myths:

The idea is to create fully bug-free software or to create a software bug-free from the start of the software development. Software bugs can be fixed at a later stage.

Methodology Myths:

A single software development method can be used for all the software development, and software development progresses linearly.

Maintenance Myths:

"No Maintenance required": Thinking that software development is complete once the initial version is released, with no further updates or maintenance needed.

"Old Software Is Obsolete": The belief that older software is always inferior to new software.

Estimation Myths:

"Cost Predictability": Thinking that the cost of a software project can be accurately predicted from the outset.

"Fixed Schedule": Believing that a project can adhere to a rigid schedule without adjustments.

Risk Myths:

"Zero Risk Software": Assuming that it's possible to develop software with absolutely no risks.

How to Avoid Software Myths

Here are some steps you can take to avoid falling victim to common software myths:

o **Stay Informed:** Keep up to date with the latest trends, practices, and developments in the field of software engineering. Attend conferences, read industry publications, and participate in online communities to stay informed.

o **Continuous Learning:** Invest in ongoing education and professional development. Software engineering is an evolving field, and staying current is essential.

o **Data-Driven Decision-Making:** Make decisions based on data, evidence, and real-world experiences rather than relying on anecdotal evidence or common misconceptions.

o **Testing and Validation:** Don't rely solely on assumptions. Test your software, gather data, and validate your ideas to confirm their accuracy.

o **Educate Team Members:** Ensure that everyone on your development team is aware of common software myths and is committed to avoiding them. Knowledge sharing and education can help dispel misconceptions.

o **Risk Assessment:** When dealing with software development decisions, conduct risk assessments to identify potential pitfalls and myths that might affect the project.