BCA V Semester

BCA-S501 Software Engineering

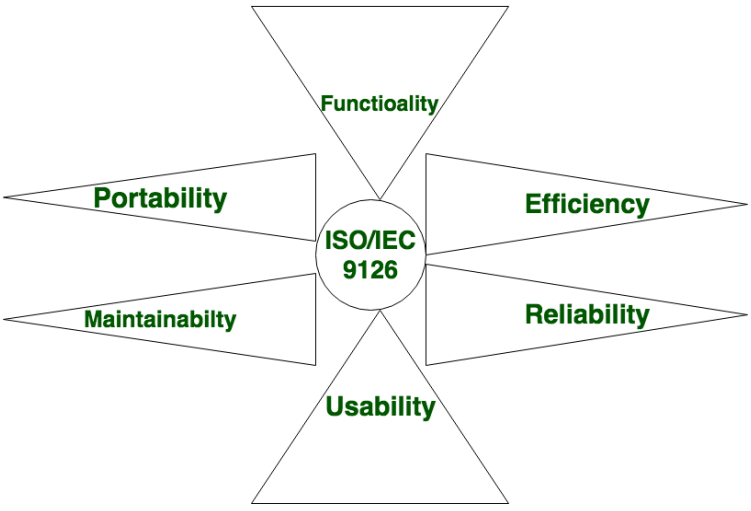
Unit-1

Software Engineering Fundamentals:

Definition of Software: 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.

Software Characteristics: **Software** is defined as a collection of computer programs, procedures, rules, and data. Software Characteristics are classified into six major components:
Software engineering is the process of designing, developing, testing, and maintaining software. The characteristics of software include:

1. It is intangible, meaning it cannot be seen or touched.
2. It is non-perishable, meaning it does not degrade over time.
3. It is easy to replicate, meaning it can be copied and distributed easily.
4. It can be complex, meaning it can have many interrelated parts and features.
5. It can be difficult to understand and modify, especially for large and complex systems.
6. It can be affected by changing requirements, meaning it may need to be updated or modified as the needs of users change.
7. It can be affected by bugs and other issues, meaning it may need to be tested and debugged to ensure it works as intended.

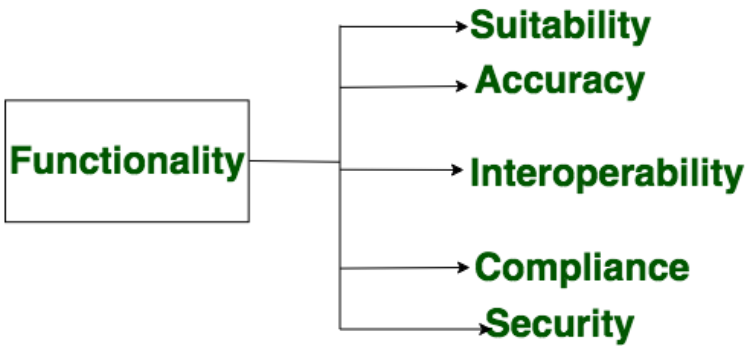


These components are described below:

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:

1. Data storage and retrieval
2. Data processing and manipulation
3. User interface and navigation
4. Communication and networking
5. Security and access control
6. Reporting and visualization
7. Automation and scripting

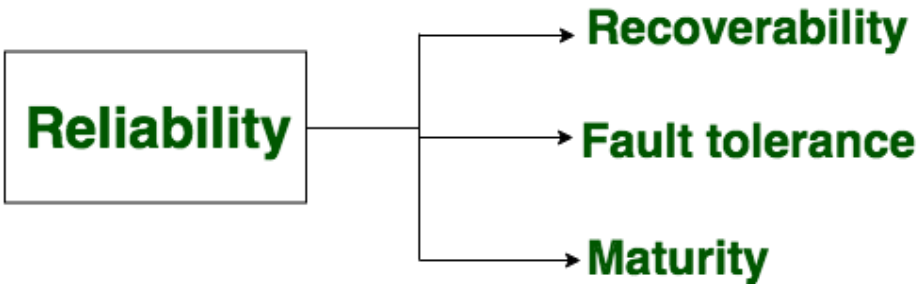


Reliability:
A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time. Reliability is a characteristic of software that refers to its ability to perform its intended functions correctly and consistently over 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. A 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:



- **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.

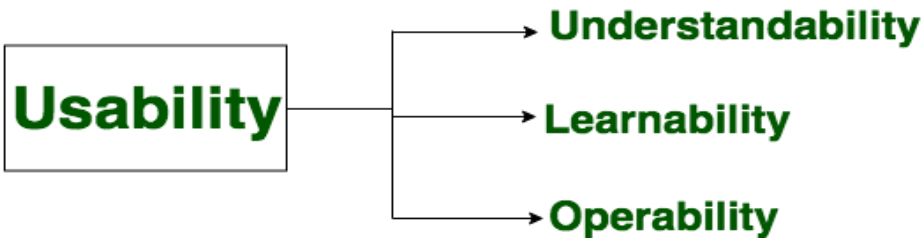
Efficiency is a characteristic of software that refers to its ability to use resources such as memory, processing power, and network bandwidth in an optimal way. High efficiency means that a software program can perform its intended functions quickly and with minimal use of resources, while low efficiency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

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

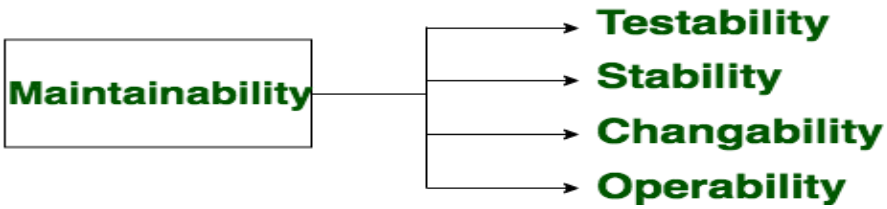
1. Poorly designed algorithms and data structures
2. Inefficient use of memory and processing power
3. High network latency or bandwidth usage
4. Unnecessary processing or computation
5. Un optimized code

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

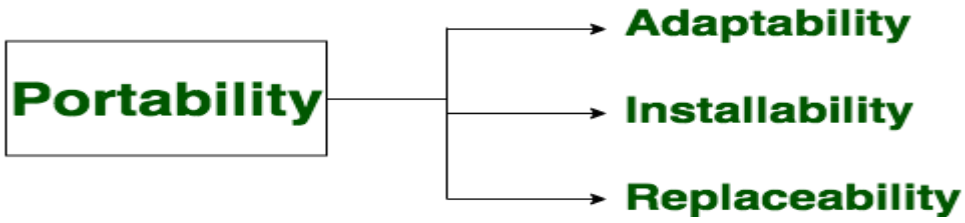
Efficiency is important in software systems that are resource-constrained, high-performance, and real-time systems. It is also important in systems that need to handle a large number of users or transactions simultaneously.
Required functions are:



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



Portability:
A set of attributes that bears on the ability of software to be transferred from one environment to another, without or minimum changes.
Required functions are:



Apart from above mention qualities of software, there are various characteristics of software in software engineering:

- **Software is developed or engineered; it is not manufactured in the classical sense:**
  - Although some similarities exist between software development and hardware manufacturing, few activities are fundamentally different.
  - In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems than software.
- **The software doesn't "wear out."**
  - Hardware components suffer from the growing effects of many other environmental factors. Stated simply, the hardware begins to wear out.
  - Software is not susceptible to the environmental maladies that cause hardware to wear out.
  - When a hardware component wears out, it is replaced by a spare part.
  - There are no software spare parts.
  - Every software failure indicates an error in design or in the process through which design was translated into machine-executable code. Therefore, the software maintenance tasks that accommodate requests for change involve considerably more complexity than hardware maintenance. However, the implication is clear—the software doesn't wear out. But it does deteriorate.
- **The software continues to be custom-built:**
  - A software part should be planned and carried out with the goal that it tends to be reused in various projects.
  - Current reusable segments encapsulate the information and the preparation that is applied to the information, empowering the programmer to make new applications from reusable parts.
  - In the hardware world, component reuse is a natural part of the engineering process.

The software is used extensively in several domains including hospitals, banks, schools, defence, finance, stock markets, and so on. It can be categorized into different types:

On the basis of application:

1. System Software –

System Software is necessary to manage the computer resources and support the execution of application programs. Software like operating systems, compilers, editors and drivers, etc., comes 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 fulfil the user's requirement by interacting with the user directly. It could be classified into two major categories:- generic or customized. Generic Software is the 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 that are designed as 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. comes 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. Utilities 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 anti-virus software, voice recognition software, compression programs, etc.

11. Document Management Software –
    Document Management Software is used to track, manage and store documents in order 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.

On the basis of copyright:

Commercial –
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 copyright of the program is owned by the company.

Shareware –
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.

Freeware –
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.

Public Domain –
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 Process: Software processes in software engineering refer to the methods and techniques used to develop and maintain software.
Software is the set of instructions in the form of programs to govern the computer system and to process the hardware components. To produce a software product the set of activities is used. This set is called a software process.
Software Development: In this process, designing, programming, documenting, testing, and bug fixing is done.

Components of Software:

There are three components of the software: These are: Program, Documentation, and Operating Procedures.

1. **Program –**
   A computer program is a list of instructions that tell a computer what to do.

2. **Documentation –**
   Source information about the product contained in design documents, detailed code comments, etc.

3. **Operating Procedures –**
   Set of step-by-step instructions compiled by an organization to help workers carry out complex routine operations.
4. **Code:** the instructions that a computer executes in order to perform a specific task or set of tasks.
5. **Data:** the information that the software uses or manipulates.
6. **User interface:** the means by which the user interacts with the software, such as buttons, menus, and text fields.
7. **Libraries:** pre-written code that can be reused by the software to perform common tasks.
8. **Documentation:** information that explains how to use and maintain the software, such as user manuals and technical guides.
9. **Test cases:** a set of inputs, execution conditions, and expected outputs that are used to test the software for correctness and reliability.
10. **Configuration files:** files that contain settings and parameters that are used to configure the software to run in a specific environment**.**
11. **Build and deployment scripts:** scripts or tools that are used to build, package, and deploy the software to different environments.
12. **Metadata:** information about the software, such as version numbers, authors, and copyright information.
13. All these components are important for software development, testing and deployment.

There are four basic key process activities:

1. **Software Specifications –**
   In this process, detailed description of a software system to be developed with its functional and non-functional requirements.
2. **Software Development –**
   In this process, designing, programming, documenting, testing, and bug fixing is done.
3. **Software Validation –**
   In this process, evaluation software product is done to ensure that the software meets the business requirements as well as the end user's needs.
4. **Software Evolution –**
   It is a process of developing software initially, then timely updating it for various reasons.

Software Crisis:

1. **Size and Cost –**
   Day to day growing complexity and expectation out of software. Software are more expensive and more complex.

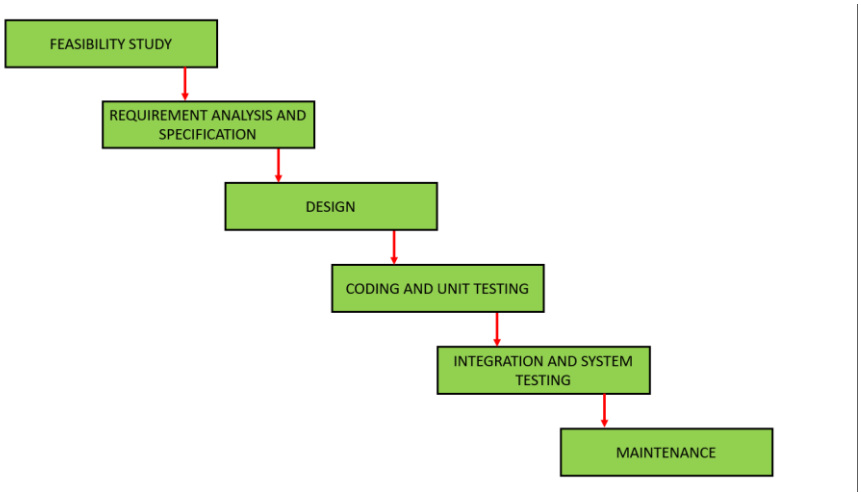
2. **Quality –**
   Software products must have good quality.

3. **Delayed Delivery –**
   Software takes longer than the estimated time to develop, which in turn leads to cost shooting up.
4. The term “software crisis” refers to a set of problems that were faced by the software industry in the 1960s and 1970s, such as:
5. High costs and long development times: software projects were taking much longer and costing much more than expected.
   6. **Low quality**: software was often delivered late, with bugs and other defects that made it difficult to use.
   7. **Lack of standardization**: there were no established best practices or standards for software development, making it difficult to compare and improve different approaches.
   8. **Lack of tools and methodologies**: there were few tools and methodologies available to help with software development, making it a difficult and time-consuming process.
   9. These problems led to a growing realization that the traditional approaches to software development were not effective and needed to be improved. This led to the development of new software development methodologies, such as the Waterfall and Agile methodologies, as well as the creation of new tools and technologies to support software development.

Software Process Model

1. **Classical Waterfall Model:**
   The classical waterfall model is the basic software development life cycle model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.

   The classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after the completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure:



1. **Feasibility Study**:
   The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software.
   The feasibility study involves understanding the problem and then determining the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.

2. **Requirements analysis and specification**:
   The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.
   - **Requirement gathering and analysis:**
     Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (an inconsistent requirement is one in which some part of the requirement contradicts some other part).
   - **Requirement specification:**
     These analysed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

3. **Design**:
   The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture. A Software Design Document is used to document all of this effort (SDD)

4. **Coding and Unit testing**:
   In the coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

5. **Integration and System testing**:

Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.
System testing consists of three different kinds of testing activities as described below:

- **Alpha testing:** Alpha testing is the system testing performed by the development team.
- **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
- **Acceptance testing:** After the software has been delivered, the customer performed acceptance testing to determine whether to accept the delivered software or reject it.

2. **Maintenance:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop full software. There are basically three types of maintenance :
   - **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
   - **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.
   - **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as working on a new computer platform or with a new operating system.

Advantages of Classical Waterfall Model
The classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model:

- This model is very simple and is easy to understand.
- Phases in this model are processed one at a time.
- Each stage in the model is clearly defined.
- This model has very clear and well-understood milestones.
- Process, actions and results are very well documented.
- Reinforces good habits: define-before- design, design-before-code.
- This model works well for smaller projects and projects where requirements are well understood.

Drawbacks of Classical Waterfall Model
The classical waterfall model suffers from various shortcomings, basically, we can't use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

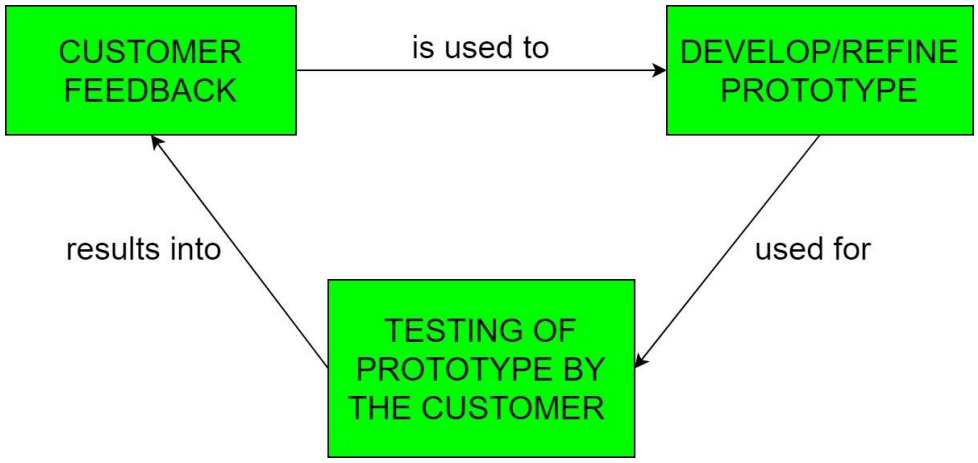
- **No feedback path:** In the classical waterfall model evolution of software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phase. Therefore, it does not incorporate any mechanism for

error correction.

- **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers' requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.

- **No overlapping of phases:** This model recommends that a new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase efficiency and reduce cost, phases may overlap.

Prototyping Model: Prototyping is defined as the process of developing a working replication of a product or system that has to be engineered. It offers a small scale facsimile of the end product and is used for obtaining customer feedback as described below:



The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.

In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle. The process starts by interviewing the customers and developing the incomplete high-level paper model. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer. Once the customer figures out the problems, the prototype is further refined to eliminate them. The process continues until the user approves the prototype and finds the working model to be satisfactory.

There are four types of models available:

A) Rapid Throwaway Prototyping –

This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of better quality.

B) Evolutionary Prototyping –

In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.

C) Incremental Prototyping –

In this type of incremental Prototyping, the final expected product is broken into different small pieces of prototypes and being developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order. It's a very efficient approach that reduces the complexity of the development process, where the goal is divided into sub-parts and each sub-part is developed individually. The time interval between the projects beginning and final delivery is substantially reduced because all parts of the system are prototyped and tested simultaneously. Of course, there might be the possibility that the pieces just do not fit together due to some lack of ness in the development phase – this can only be fixed by careful and complete plotting of the entire system before prototyping starts.

Advantages –

- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
- The developed prototype can be reused by the developer for more complicated projects in the future.

- Flexibility in design.

Disadvantages –

- There may be too much variation in requirements each time the prototype is evaluated by the customer.
- Poor Documentation due to continuously changing customer requirements.
- It is very difficult for developers to accommodate all the changes demanded by the customer.
- There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.
- After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.
- Developers in a hurry to build prototypes may end up with sub-optimal solutions.
- The customer might lose interest in the product if he/she is not satisfied with the initial prototype.

Use –
The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable. It can also be used if requirements are changing quickly. This model can be successfully used for developing user interfaces, high technology software-intensive systems, and systems with complex algorithms and interfaces. It is also a very good choice to demonstrate the technical feasibility of the product.

Spiral Model-

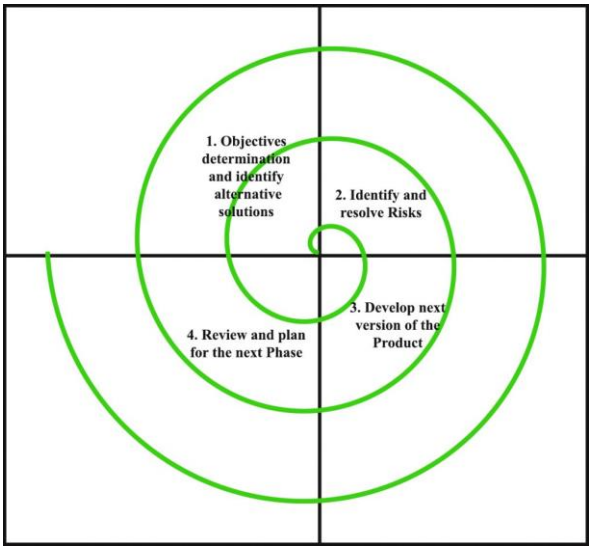
Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.

The Spiral Model is a software development life cycle (SDLC) model that provides a systematic and iterative approach to software development. It is based on the idea of a spiral, with each iteration of the spiral representing a complete software development cycle, from requirements gathering and analysis to design, implementation, testing, and maintenance.

The Spiral Model is a risk-driven model, meaning that the focus is on managing risk through multiple iterations of the software development process. It consists of the following phases:

1. Planning: The first phase of the Spiral Model is the planning phase, where the scope of the project is determined and a plan is created for the next iteration of the spiral.
2. Risk Analysis: In the risk analysis phase, the risks associated with the project are identified and evaluated.
3. Engineering: In the engineering phase, the software is developed based on the requirements gathered in the previous iteration.
4. Evaluation: In the evaluation phase, the software is evaluated to determine if it meets the customer's requirements and if it is of high quality.
5. Planning: The next iteration of the spiral begins with a new planning phase, based on the results of the evaluation.
6. The Spiral Model is often used for complex and large software development projects, as it allows for a more flexible and adaptable approach to software development. It is also well-suited to projects with significant uncertainty or high levels of risk.
   The Radius of the spiral at any point represents the expenses (cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.

The below diagram shows the different phases of the Spiral Model: –



Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Why Spiral Model is called Meta Model?

The Spiral model is called a Meta-Model because it subsumes all the other SDLC models. For example, a single loop spiral actually represents the Iterative Waterfall Model. The spiral model incorporates the stepwise approach of the Classical Waterfall Model. The spiral model uses the approach of the Prototyping Model by building a prototype at the start of each phase as a risk-handling technique. Also, the spiral model can be considered as supporting the Evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

Advantages of Spiral Model:

Below are some advantages of the Spiral Model.

1. **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
2. **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
3. **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
4. **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.
5. Iterative and Incremental Approach: The Spiral Model provides an iterative and incremental approach to software development, allowing for flexibility and adaptability in response to changing requirements or unexpected events.
6. Emphasis on Risk Management: The Spiral Model places a strong emphasis on risk management, which helps to minimize the impact of uncertainty and risk on the software development process.
7. Improved Communication: The Spiral Model provides for regular evaluations and reviews, which can improve communication between the customer and the development team.

8. Improved Quality: The Spiral Model allows for multiple iterations of the software development process, which can result in improved software quality and reliability

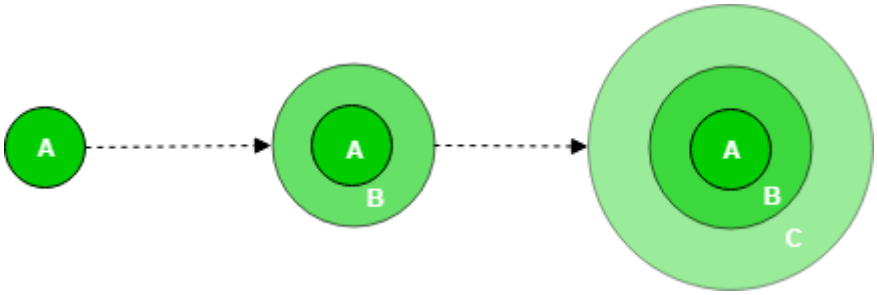
Disadvantages of Spiral Model:

Below are some main disadvantages of the spiral model.

1. **Complex:** The Spiral Model is much more complex than other SDLC models.
2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.
5. **Complexity**: The Spiral Model can be complex, as it involves multiple iterations of the software development process.
6. **Time-Consuming**: The Spiral Model can be time-consuming, as it requires multiple evaluations and reviews.
7. **Resource Intensive**: The Spiral Model can be resource-intensive, as it requires a significant investment in planning, risk analysis, and evaluations

Incremental Model:

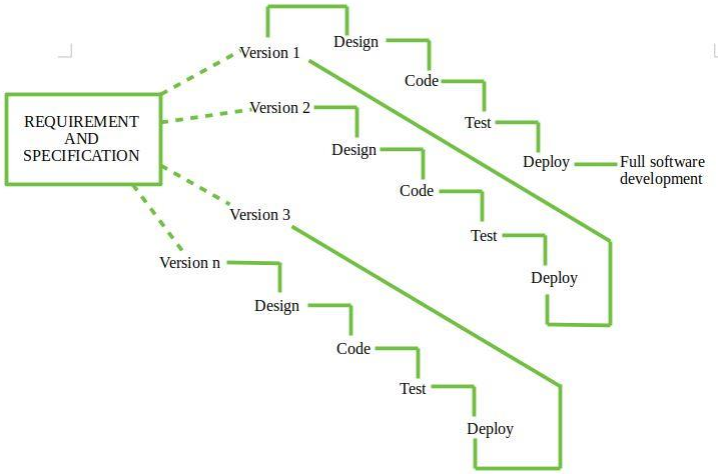
The incremental process model is also known as the Successive version model.
First, a simple working system implementing only a few basic features is built and then that is delivered to the customer. Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.



A, B, and C are modules of Software Products that are incrementally developed and delivered.

Life cycle activities:

Requirements of Software are first broken down into several modules that can be incrementally constructed and delivered. At any time, the plan is made just for the next increment and not for any kind of long-term plan. Therefore, it is easier to modify the version as per the need of the customer. The Development Team first undertakes to develop core features (these do not need services from other features) of the system.
Once the core features are fully developed, then these are refined to increase levels of capabilities by adding new functions in Successive versions. Each incremental version is usually developed using an iterative waterfall model of development.
As each successive version of the software is constructed and delivered, now the feedback of the Customer is to be taken and these were then incorporated into the next version. Each version of the software has more additional features than the previous ones.



After Requirements gathering and specification, requirements are then split into several different versions starting with version 1, in each successive increment, the next version is constructed and then deployed at the customer site. After the last version (version n), it is now deployed at the client site.

When to use this:

1. Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.

2. When Requirements are known up-front.

3. When Projects have lengthy development schedules.

4. Projects with new Technology.

- Error Reduction (core modules are used by the customer from the beginning of the phase and then these are tested thoroughly)
- Uses divide and conquer for a breakdown of tasks.
- Lowers initial delivery cost.
- Incremental Resource Deployment.

5. Requires good planning and design.

6. The total cost is not lower.

7. Well-defined module interfaces are required.

Advantages-

1. Prepares the software fast.
2. Clients have a clear idea of the project.
3. Changes are easy to implement.
4. Provides risk handling support, because of its iterations.

Disadvantages-

1. A good team and proper planned execution are required.
2. Because of its continuous iterations the cost increases.

Concurrent Development Model-

The concurrent development model, sometimes called concurrent engineering.

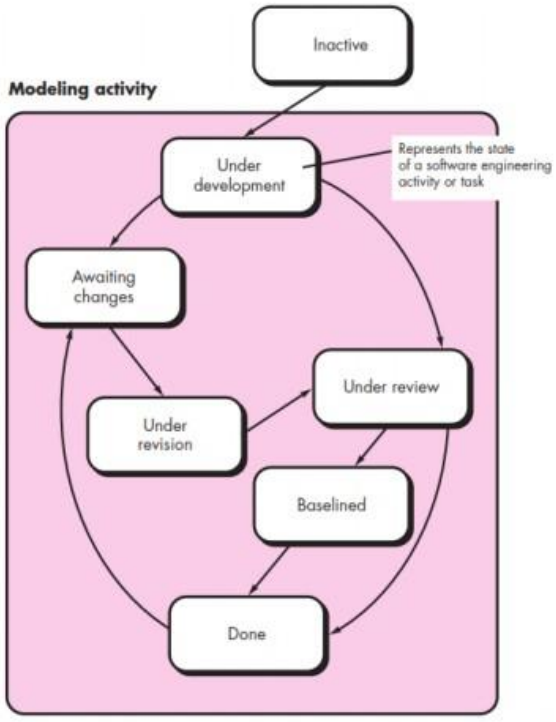


Figure: Concurrent Process Model

- It allows a software team to represent iterative and concurrent elements of any of the process model.

- For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the software engineering actions: prototyping, analysis, and design.

- The activity—modelling—may be in any one of the states noted at any given time.

- Similarly, other activities, actions, or tasks (e.g., communication or construction) can be represented in similar manner.
- All software engineering activities exist concurrently but reside in different states.

- For example, early in a project the communication activity (not shown in the figure) has completed its first iteration and exists in the awaiting changes state.

The modelling activity (which existed in the inactive state while initial communication was completed, now makes a transition into the under development state. If, however, the customer indicates that changes in requirements must be made, the modelling activity moves from the under development state into the awaiting changes state.
Concurrent modelling defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks.

Project Management Concept-
What is Project?

A project is a group of tasks that need to complete to reach a clear result. A project also defines as a set of inputs and outputs which are required to achieve a goal. Projects can vary from simple to difficult and can be operated by one person or a hundred.

Projects usually described and approved by a project manager or team executive. They go beyond their expectations and objects, and it's up to the team to handle logistics and complete the project on time. For good project development, some teams split the project into specific tasks so they can manage responsibility and utilize team strengths.

What is software project management?

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled.

It is a procedure of managing, allocating and timing resources to develop computer software that fulfils requirements.

In software Project Management, the client and the developers need to know the length, period and cost of the project.

There are three needs for software project management. These are:

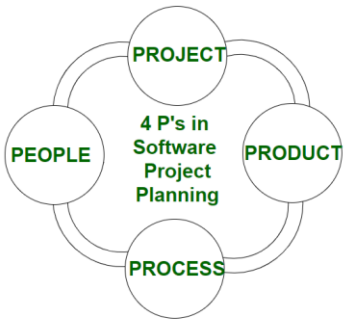
1. Time
2. Cost
3. Quality

It is an essential part of the software organization to deliver a quality product, keeping the cost within the client's budget and deliver the project as per schedule. There are various factors, both external and internal, which may impact this triple factor. Any of three-factor can severely affects the other two.

The Management Spectrum

For properly building a product, there's a very important concept that we all should know in software project planning while developing a product. There are 4 critical components in software project planning which are known as the 4P's namely:

- Product
- Process
- People
- Project



These components play a very important role in your project that can help your team meet its goals and objective. Now, Let's dive into each of them a little in detail to get a better understanding:

- **People**
  The most important component of a product and its successful implementation is human resources. In building a proper product, a well-managed team with clear-cut roles defined for each person/team will lead to the success of the product. We need to have a good team in order to save our time, cost, and effort. Some assigned roles in software project planning are project manager, team leaders, stakeholders, analysts, and other IT professionals. Managing people successfully is a tricky process which a good project manager can do.

- **Product**
  As the name inferred, this is the deliverable or the result of the project. The project manager should clearly define the product scope to ensure a successful result, control the team members, as well technical hurdles that he or she may encounter during the building of a product. The product can consist of both tangible and intangible such as shifting the company to a new place or getting new software in a company

- **Process**
  In every planning, a clearly defined process is the key to the success of any product. It regulates how the team will go about its development in the respective time period. The Process has several steps involved like, documentation phase, implementation phase, deployment phase, and interaction phase.

- **Project**
  The last and final P in software project planning is Project. It can also be considered as a blueprint of process. In this phase, the project manager plays a critical role. They are responsible to guide the team members to achieve the project's target and objectives, helping & assisting them with issues, checking on cost and budget, and making sure that the project stays on track with the given deadlines.