OOSE Notes

UNIT I

Introduction to Software Engineering

- The Evolving Role Of Software
- Software
- The Changing Nature Of Software
- Legacy Software

Process Models:

- The Waterfall Model
- Incremental Process Models
- Evolutionary Process Models
- Specialized Process Models.

THE EVOLVING ROLE OF A SOFTWARE

What is 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.

THE EVOLVING ROLE OF SOFTWARE

Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation.

As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments). Software delivers the most important product of our time—information.

Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.

The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems. The lone programmer of an earlier era has been replaced by a team of software specialists, each focusing on one part of the technology required to deliver a complex application.

Changing Nature of Software:

The nature of software has changed a lot over the years.

1.System software: Infrastructure software come under this category like compilers, operating systems, editors, drivers, etc. Basically system software is a collection of programs to provide service to other programs.

2. Real time software: These software are used to monitor, control and analyze real world events as they occur. An example may be software required for weather forecasting. Such software will gather and process the status of temperature, humidity and other environmental parameters to forcast the weather.

3. Embedded software: This type of software is placed in "Read-Only- Memory (ROM)"of the product and control the various functions of the product. The product could be an aircraft, automobile, security system, signalling system, control unit of power plants, etc. he embedded software handles hardware components and is also termed as intelligent software .

4. Business software : This is the largest application area. The software designed to process business applications is called business software. Business software could be payroll, file monitoring system, employee management, account management. It may also be a data warehousing tool which helps us to take decisions based on available data. Management information system, enterprise resource planning (ERP) and such other software are popular examples of business software.

5. Personal computer software: The software used in personal computers are covered in this category. Examples are word processors, computer graphics, multimedia and animating tools, database management, computer games etc. This is a very upcoming area and many big organisations are concentrating their effort here due to large customer base.

6. Artificial intelligence software: Artificial Intelligence software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis. Examples are expert systems, artificial neural network,signal processing software etc.

7. Web based software: The software related to web applications come under this category. Examples are CGI, HTML, Java, Perl, DHTML etc.

LEGACY SOFTWARE:

Legacy software is **software** that has been around a long time and still fulfills a business need. It is mission critical and tied to a particular version of an operating system or hardware model (vendor lock-in) that has gone end-of-life. Generally the lifespan of the hardware is shorter than that of the **software**.

PROCESS MODELS:

Software Processes is a coherent set of activities for specifying, designing, implementing and testing software systems. A software process model is an abstract representation of a process that presents a description of a process from some particular perspective. There are many different software processes but all involve:

- Specification – defining what the system should do;
- Design and implementation – defining the organization of the system and implementing the system;
- Validation – checking that it does what the customer wants;
- Evolution – changing the system in response to changing customer needs.

Types of Software Process Model

Software processes, methodologies and frameworks range from specific prescriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the

needs of a specific project or group. In some cases a "sponsor" or "maintenance" organization distributes an official set of documents that describe the process.

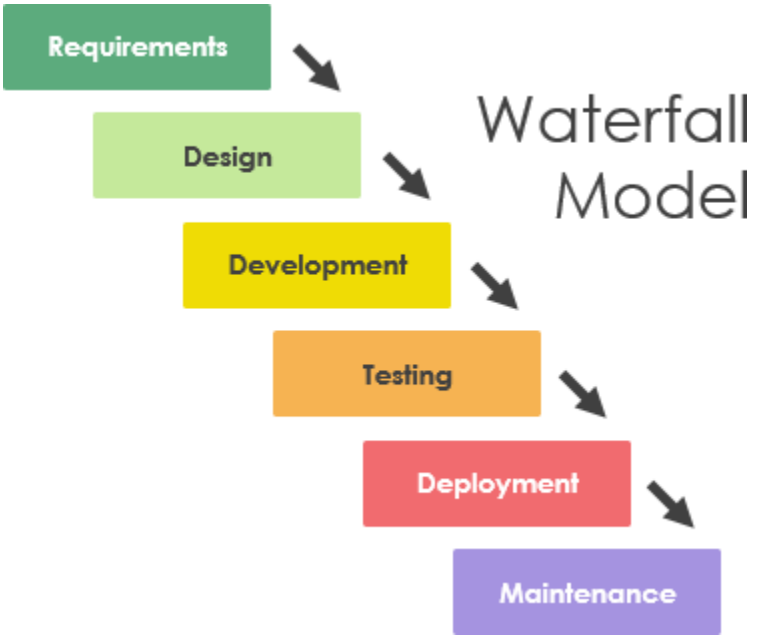
Software Process and Software Development Lifecycle Model

One of the basic notions of the software development process is SDLC models which stands for Software Development Life Cycle models. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out. The most used, popular and important SDLC models are given below:

- Waterfall model
- V model
- Incremental model
- RAD model
- Agile model
- Iterative model
- Spiral model
- Prototype model

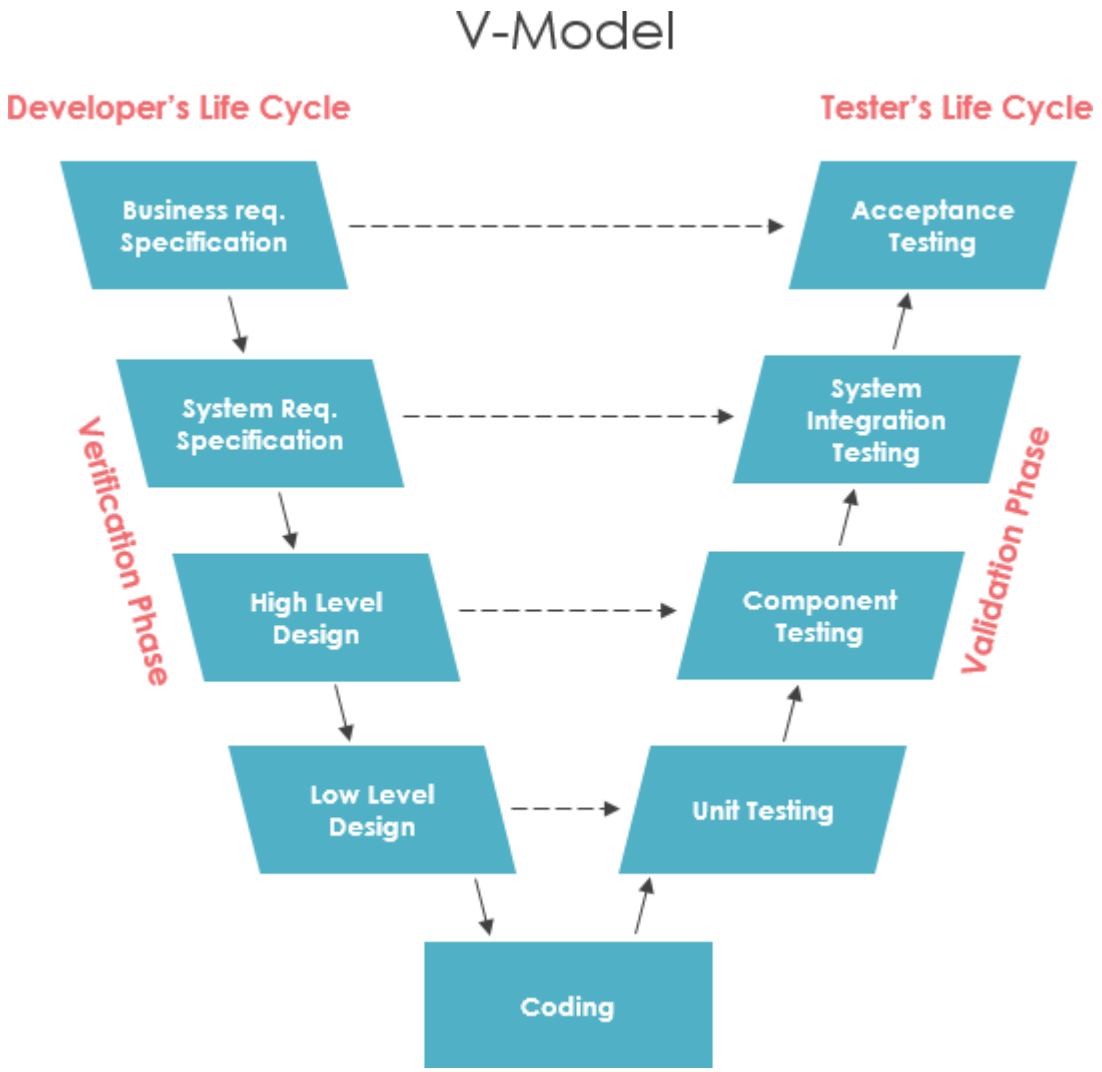
Waterfall Model

The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialisation of tasks. The approach is typical for certain areas of engineering design.



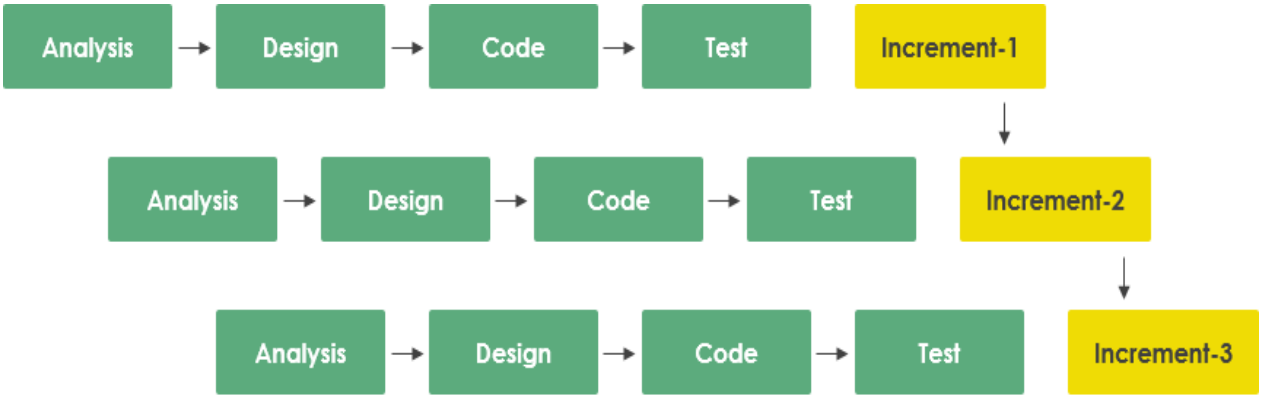
V Model

The V-model represents a development process that may be considered an extension of the waterfall model and is an example of the more general V-model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.



Incremental model

The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements. Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionally has been implemented. This model combines the elements of the waterfall model with the iterative philosophy of prototyping.

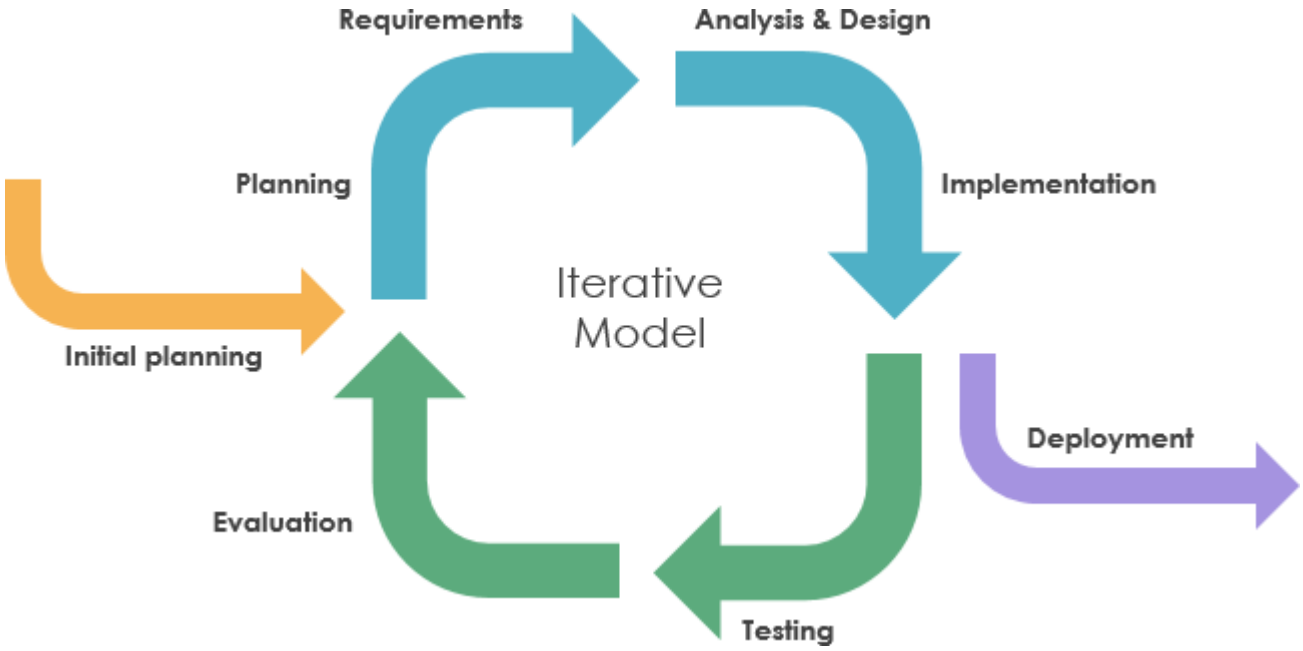


Incremental Model

Iterative Model

An iterative life cycle model does not attempt to start with a full specification of requirements by first focusing on an initial, simplified set user features, which then progressively gains more complexity and a broader set of features until the targeted system is complete. When adopting the iterative approach, the philosophy of incremental development will also often be used liberally and interchangeably.

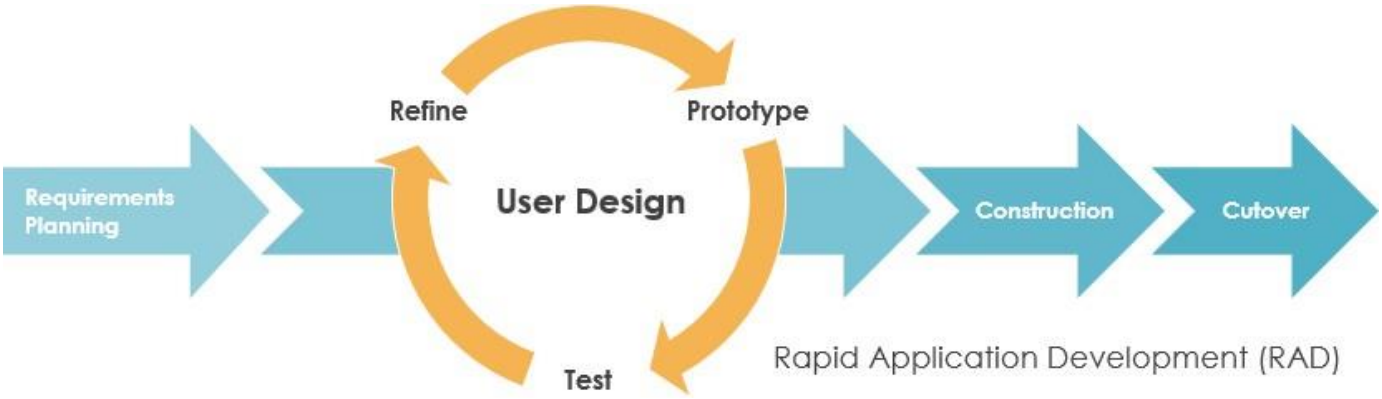
In other words, the iterative approach begins by specifying and implementing just part of the software, which can then be reviewed and prioritized in order to identify further requirements. This iterative process is then repeated by delivering a new version of the software for each iteration. In a light-weight iterative project the code may represent the major source of documentation of the system; however, in a critical iterative project a formal software specification may also be required.



RAD model

Rapid application development was a response to plan-driven waterfall processes, developed in the 1970s and 1980s, such as the Structured Systems Analysis and Design Method (SSADM). Rapid application development (RAD) is often referred as the adaptive software development. RAD is an incremental prototyping approach to software development that end users can produce better feedback when examining a live system, as opposed to working strictly with documentation. It puts less emphasis on planning and more emphasis on an adaptive process.

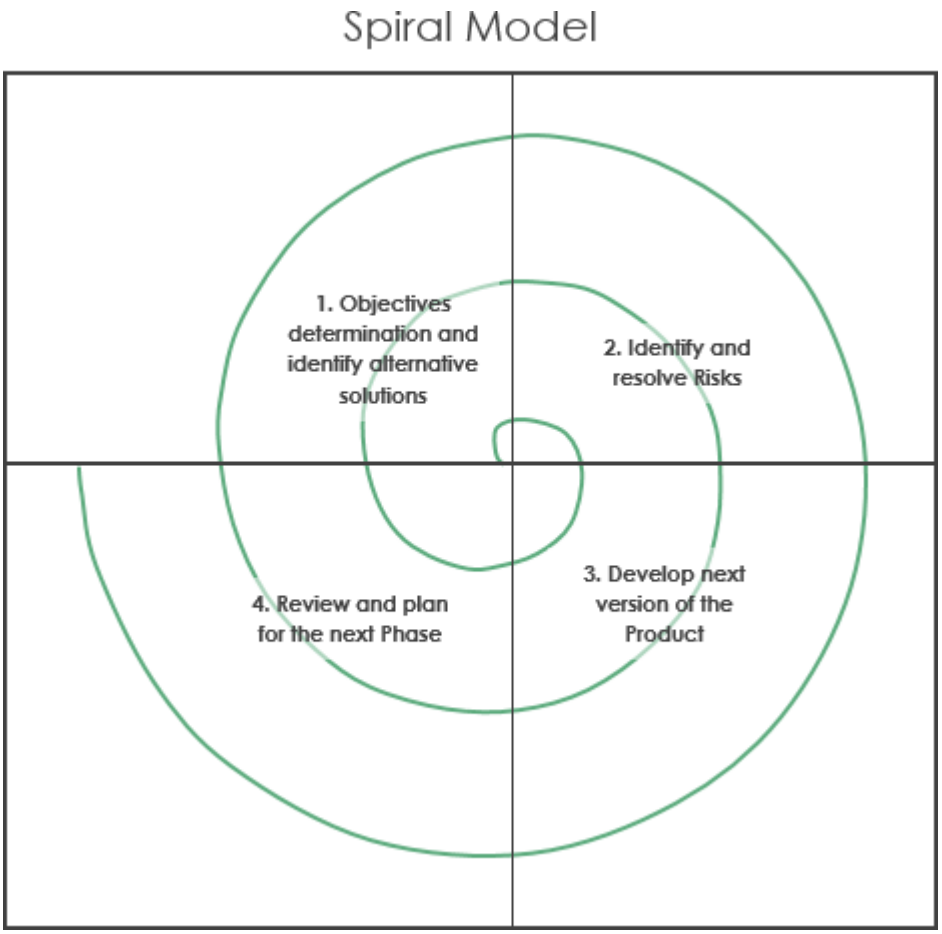
RAD may resulted in a lower level of rejection when the application is placed into production, but this success most often comes at the expense of a dramatic overruns in project costs and schedule. RAD approach is especially well suited for developing software that is driven by user interface requirements. Thus, some GUI builders are often called rapid application development tools.



Spiral model

The spiral model, first described by Barry Boehm in 1986, is a risk-driven software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model. A spiral model looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. This model supports risk handling, and the project is delivered in loops. Each loop of the spiral is called a Phase of the software development process.

The initial phase of the spiral model in the early stages of Waterfall Life Cycle that is needed to develop a software product. 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 a spiral model.



Evolutionary Process Models

- Evolutionary models are iterative type models.
- They allow to develop more complete versions of the software.

Following are the evolutionary process models.

1. The prototyping model
2. The spiral model
3. Concurrent development model

1. The Prototyping model

- Prototype is defined as first or preliminary form using which other forms are copied or derived.
- Prototype model is a set of general objectives for software.
- It does not identify the requirements like detailed input, output.
- It is software working model of limited functionality.
- In this model, working programs are quickly produced.

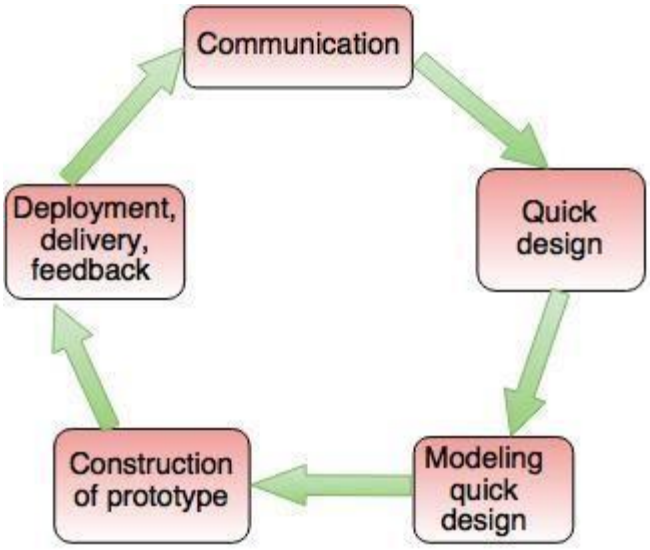


Fig. - The Prototyping Model

The different phases of Prototyping model are:

1.Communication

In this phase, developer and customer meet and discuss the overall objectives of the software.

2. Quick design

- Quick design is implemented when requirements are known.
- It includes only the important aspects like input and output format of the software.
- It focuses on those aspects which are visible to the user rather than the detailed plan.

- It helps to construct a prototype.

3. Modeling quick design

- This phase gives the clear idea about the development of software because the software is now built.
- It allows the developer to better understand the exact requirements.

4.Constructionof prototype

The prototype is evaluated by the customer itself.

5. Deployment, delivery, feedback

- If the user is not satisfied with current prototype then it refines according to the requirements of the user.
- The process of refining the prototype is repeated until all the requirements of users are met.
- When the users are satisfied with the developed prototype then the system is developed on the basis of final prototype.

Advantages of Prototyping Model

- Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.
- Gives quick user feedback for better solutions.
- It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

Disadvantages of Prototyping Model:

- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.
- It is a thrown away prototype when the users are confused with it.

2. The Spiral model

- Spiral model is a risk driven process model.
- It is used for generating the software projects.
- In spiral model, an alternate solution is provided if the risk is found in the risk analysis, then alternate solutions are suggested and implemented.
- It is a combination of prototype and sequential model or waterfall model.
- In one iteration all activities are done, for large project's the output is small.

The framework activities of the spiral model are as shown in the following figure.

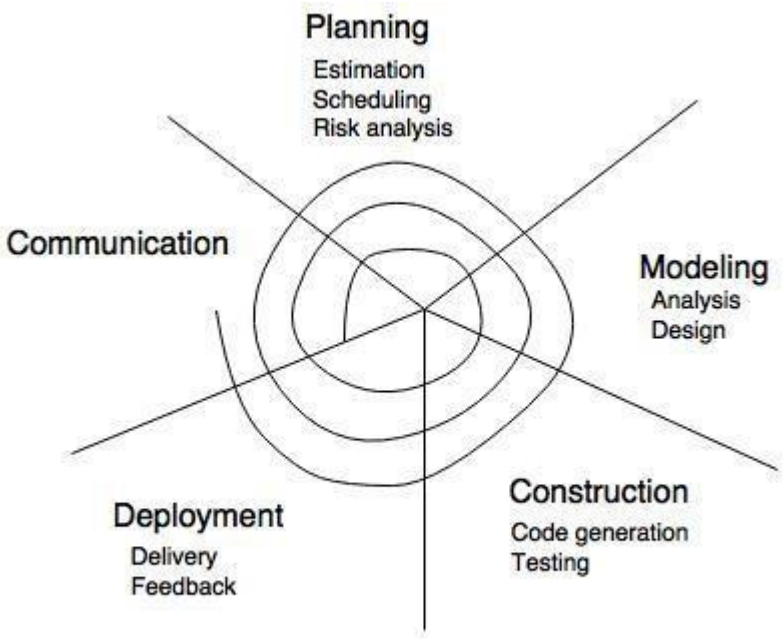


Fig. - The Spiral Model

NOTE: The description of the phases of the spiral model is same as that of the process model.

Advantages of Spiral Model

- It reduces high amount of risk.
- It is good for large and critical projects.
- It gives strong approval and documentation control.

- In spiral model, the software is produced early in the life cycle process.

Disadvantages of Spiral Model

- It can be costly to develop a software model.
- It is not used for small projects.

The concurrent development model

- The concurrent development model is called as concurrent model.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modeling activity completed its initial communication and then go to the underdevelopment state.
- If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.
- The concurrent process model activities moving from one state to another state.

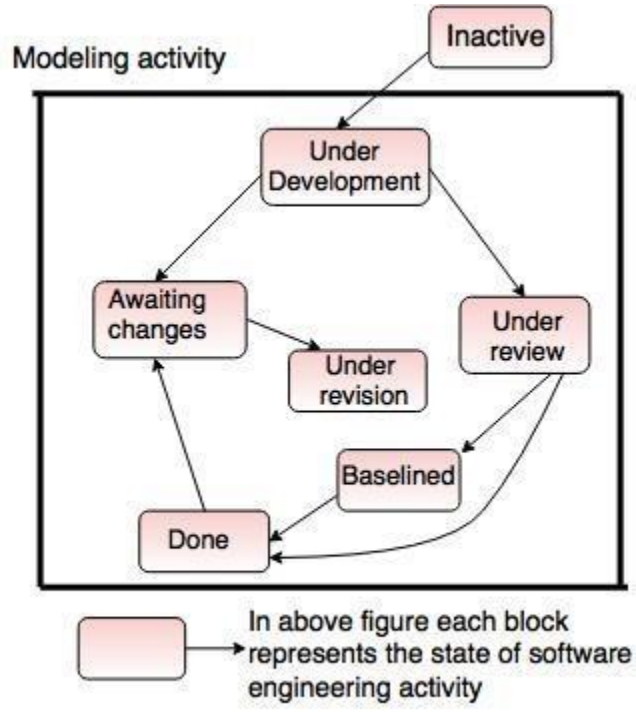


Fig. - One element of the concurrent process model

Advantages of the concurrent development model

- This model is applicable to all types of software development processes.
- It is easy for understanding and use.
- It gives immediate feedback from testing.
- It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development model

- It needs better communication between the team members. This may not be achieved all the time.
- It requires to remember the status of the different activities.

SPECIALIZED PROCESS MODELS

Special process models take many features from one or more conventional models. However these special models tend to be applied when a narrowly defined software engineering approach is chosen.

Types in Specialized process models:

1. Component based development (Promotes reusable components)
2. The formal methods model (Mathematical formal methods are backbone here)
3. Aspect oriented software development (Uses crosscutting technology)

COMPONENT BASED DEVELOPMENT

What is a Component?

A component is a modular, portable, replaceable, and reusable set of well-defined functionality that encapsulates its implementation and exporting it as a higher-level interface.

A component is a software object, intended to interact with other components, encapsulating certain functionality or a set of functionalities. It has an obviously defined

interface and conforms to a recommended behavior common to all components within an architecture.

A software component can be defined as a unit of composition with a contractually specified interface and explicit context dependencies only. That is, a software component can be deployed independently and is subject to composition by third parties.

Views of a Component

A component can have three different views – object-oriented view, conventional view, and process-related view.

Object-oriented view

A component is viewed as a set of one or more cooperating classes. Each problem domain class (analysis) and infrastructure class (design) are explained to identify all attributes and operations that apply to its implementation. It also involves defining the interfaces that enable classes to communicate and cooperate.

Conventional view

It is viewed as a functional element or a module of a program that integrates the processing logic, the internal data structures that are required to implement the processing logic and an interface that enables the component to be invoked and data to be passed to it.

Process-related view

In this view, instead of creating each component from scratch, the system is building from existing components maintained in a library. As the software architecture is formulated, components are selected from the library and used to populate the architecture.

- A user interface (UI) component includes grids, buttons referred as controls, and utility components expose a specific subset of functions used in other components.
- Other common types of components are those that are resource intensive, not frequently accessed, and must be activated using the just-in-time (JIT) approach.

- Many components are invisible which are distributed in enterprise business applications and internet web applications such as Enterprise JavaBean (EJB), .NET components, and CORBA components.

Characteristics of Components

- **Reusability** – Components are usually designed to be reused in different situations in different applications. However, some components may be designed for a specific task.
- **Replaceable** – Components may be freely substituted with other similar components.
- **Not context specific** – Components are designed to operate in different environments and contexts.
- **Extensible** – A component can be extended from existing components to provide new behavior.
- **Encapsulated** – A A component depicts the interfaces, which allow the caller to use its functionality, and do not expose details of the internal processes or any internal variables or state.
- **Independent** – Components are designed to have minimal dependencies on other components.

FORMAL METHODS MODEL

The **formal methods model** is concerned with the application of a mathematical technique to design and implement the software. This model lays the foundation for developing a complex system and supporting the program development. The formal methods used during the development process provide a mechanism for eliminating problems, which are difficult to overcome using other software process models. The software engineer creates formal specifications for this model. These methods minimize specification errors and this result in fewer errors when the user begins using the system.

Table Advantages and Disadvantages of Formal Methods Model

Advantages	Disadvantages
▪ Discovers ambiguity, incompleteness, and inconsistency in the software. ▪ Offers defect-free software. ▪ Incrementally grows in effective solution after each iteration. ▪ This model does not involve high complexity rate. ▪ Formal specification language semantics verify self-consistency.	▪ Time consuming and expensive. ▪ Difficult to use this model as a communication mechanism for non technical personnel. ▪ Extensive training is required since only few developers have the essential knowledge to implement this model.

ASPECT ORIENTED SOFTWARE DEVELOPMENT

Aspect-oriented software development (AOSD) is a software programming solution built to address modularity deficiencies of traditional software development approaches such as structural, procedural and object-oriented programming methods. It is an approach that is designed to complement the conventional designs rather than replace them.

Features Of Aspect-Oriented Software Development

* AOSD focuses on core areas of identification and investigation, specification and representation of concerns that can present crosscutting concerns
* AOSD gives complementary benefits and can be used along with other coding standards.
* AOSD ensures better modularization mechanisms of program designs, thereby reducing software design, and maintenance and development costs.
* AOSD is a better way to handle localization of concerns that are cross-cutting because concerns are compressed into various modules.
* AOSD makes available software coding methods and tools to support modularization at

the source code stage.

* AOSD ensures improved and smaller size of code as a result of addressing the cross-cutting issues

* AOSD encourages the reuse of code created by the modularization technique.

UNIT II

Requirements Engineering:

- Requirements Engineering Tasks
- Initiating the Requirements Engineering Process
- Eliciting Requirements
- Developing Use-Cases.

INTRODUCTION

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly –

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various

platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

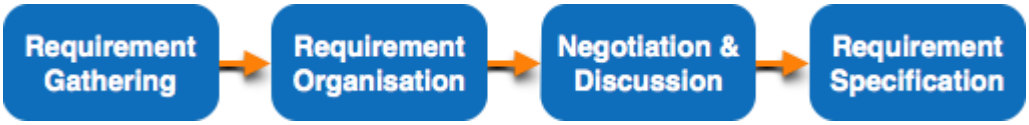
Software Requirement Validation

After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the folloiwng diagram:



- **Requirements gathering -** The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements -** The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion -** If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

  The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation -** All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Software Requirements Characteristics

Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct and well-defined.

A complete Software Requirement Specifications must be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirements are expected from the software system.

Broadly software requirements should be categorized in two categories:

Functional Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

Examples -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.

- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders and negation, whereas 'could have' and 'wish list' can be kept for software updates.

User Interface requirements

UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is -

- easy to operate
- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system can not be used in convenient way. A system is said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings

- Purposeful layout
- Strategical use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Software System Analyst

System analyst in an IT organization is a person, who analyzes the requirement of proposed system and ensures that requirements are conceived and documented properly & correctly. Role of an analyst starts during Software Analysis Phase of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- Analyzing and understanding requirements of intended software
- Understanding how the project will contribute in the organization objectives
- Identify sources of requirement
- Validation of requirement
- Develop and implement requirement management plan
- Documentation of business, technical, process and product requirements
- Coordination with clients to prioritize requirements and remove and ambiguity
- Finalizing acceptance criteria with client and other stakeholders

Software Metrics and Measures

Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software.

Software Metrics provide measures for various aspects of software process and software product.

Software measures are fundamental requirement of software engineering. They not only help to control the software development process but also aid to keep quality of ultimate product excellent.

According to Tom DeMarco, a (Software Engineer), "You cannot control what you cannot measure." By his saying, it is very clear how important software measures are.

Let us see some software metrics:

- **Size Metrics -** LOC (Lines of Code), mostly calculated in thousands of delivered source code lines, denoted as KLOC.

  Function Point Count is measure of the functionality provided by the software. Function Point count defines the size of functional aspect of software.

- **Complexity Metrics -** McCabe's Cyclomatic complexity quantifies the upper bound of the number of independent paths in a program, which is perceived as complexity of the program or its modules. It is represented in terms of graph theory concepts by using control flow graph.
- **Quality Metrics -** Defects, their types and causes, consequence, intensity of severity and their implications define the quality of product.

  The number of defects found in development process and number of defects reported by the client after the product is installed or delivered at client-end, define quality of product.

- **Process Metrics -** In various phases of SDLC, the methods and tools used, the company standards and the performance of development are software process metrics.
- **Resource Metrics -** Effort, time and various resources used, represents metrics for resource measurement.

DEVELOPING USE – CASES

Before we start working on any project, it is very important that we are very clear on what we want to do and how do we want to do. In my last Blog, I discussed on how to write a good SRS for your project and what is the advantage we get out of that. In this Blog, I'll discuss Use Cases and their advantage in our projects.

What are Use Cases?

In software and systems engineering, a use case is a list of actions or event steps, typically defining the interactions between a role (known in the Unified Modeling Language as an *actor*) and a system, to achieve a goal. The actor can be a human, an external system, or time. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. Another way to look at it is a use case describes a way in which a real-world actor interacts with the system. In a system use case you include high-level implementation decisions. System use cases can be written in both an informal manner and a formal manner.

What is the importance of Use Cases?

Use cases have been used extensively over the past few decades. The advantages of Use cases includes:

- The list of goal names provides the shortest summary of what the system will offer
- It gives an overview of the roles of each and every component in the system. It will help us in defining the role of users, administrators etc.
- It helps us in extensively defining the user's need and exploring it as to how it will work.
- It provides solutions and answers to many questions that might pop up if we start a project unplanned.

How to plan use case?

Following example will illustrate on how to plan use cases:

Use Case: What is the main objective of this use case. For eg. Adding a software component, adding certain functionality etc.

Primary Actor: Who will have the access to this use case. In the above examples, administrators will have the access.

Scope: Scope of the use case

Level: At what level the implementation of the use case be.

Flow: What will be the flow of the functionality that needs to be there. More precisely, the work flow of the use case.

Use Case Diagram

Below is a sample use case diagram which I have prepared for reference purpose for a sample project (much like Facebook). It would help us to understand the role of various actors in our project. Various actors in the below use case diagram are: **User and System.**

The main use cases are in the system and the diagram illustrates on how the actors interact with the use cases.For eg. During Sign Up, only users need to interact with the use case and not the system whereas when it comes to categorizing posts, only system would be required.

Unit III

Estimation

1. Observation of Estimation
2. The Project Planning Process
3. Software scope and feasibility
4. Resources
5. Software Project Estimation
6. Decomposition Techniques
7. Empirical estimation model
8. Estimation for object oriented projects

Estimation

- Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.
- Estimation determines how much money, effort, resources, and time it will take to build a specific system or product.

Estimation is based on –

Past Data/Past Experience

Available Documents/Knowledge

Assumptions

Identified Risks

The four basic steps in Software Project Estimation are –

- Estimate the size of the development product.
- Estimate the effort in person-months or person-hours.
- Estimate the schedule in calendar months.
- Estimate the project cost in agreed currency.

1. Observations of Estimation

- Estimation need not be a one-time task in a project
- It can take place during –

Acquiring a Project.

Planning the Project.

Execution of the Project as the need arises.

- Project scope must be understood before the estimation process begins.
- It will be helpful to have historical Project Data.
- Project metrics can provide a historical perspective and valuable input for generation of quantitative estimates.
- Planning requires technical managers and the software team to make an initial commitment as it leads to responsibility and accountability.
- Past experience can aid greatly.
- Use at least two estimation techniques to arrive at the estimates and reconcile the resulting values.
- Refer Decomposition Techniques in the next section to learn about reconciling estimates.
- Plans should be iterative and allow adjustments as time passes and more details are known.

2. Project Planning Process

- Once a project is found to be possible, computer code project managers undertake project designing.
- Project designing is undertaken and completed even before any development activity starts.
- Project designing consists of subsequent essential activities:

Estimating the subsequent attributes of the project:

Project size:

What's going to be downside quality in terms of the trouble and time needed to develop the product?

Cost:

What proportion is it reaching to value to develop the project?

Duration:

However long is it reaching to want complete development?

Effort:

What proportion effort would be required?

- ✓ The effectiveness of the following designing activities relies on the accuracy of those estimations.
- ✓ planning force and alternative resources
- ✓ workers organization and staffing plans
- ✓ Risk identification, analysis, and abatement designing
- ✓ Miscellaneous arranges like quality assurance plan, configuration, management arrange, etc.
- ✓ Precedence ordering among project planning activities:
- ✓ The different project connected estimates done by a project manager have already been mentioned.
- ✓ The below diagram shows the order during which vital project coming up with activities is also undertaken.

- ✓ It may be simply discovered that size estimation is that the 1st activity.
- ✓ It's conjointly the foremost basic parameter supported that all alternative coming up with activities square measure dispensed, alternative estimations like the estimation of effort, cost, resource, and project length also are vital elements of the project coming up with.

Sliding Window Planning:

- ✓ Project designing needs utmost care and a spotlight since commitment to unrealistic time and resource estimates end in schedule slippage.
- ✓ Schedule delays will cause client discontent and adversely have an effect on team morale.
- ✓ It will even cause project failure.
- ✓ However, project designing could be a terribly difficult activity. particularly for giant comes, it's pretty much troublesome to create correct plans.
- ✓ A region of this issue is thanks to the actual fact that the correct parameters, the scope of the project, project workers, etc.
- ✓ might amendment throughout the span of the project. So as to beat this drawback, generally project managers undertake project designing little by little.
- ✓ Designing a project over a variety of stages protects managers from creating huge commitments too early.
- ✓ This method of staggered designing is thought of as window

designing. Within the window technique, beginning with

associate initial set up, the project is planned additional accurately in sequential development stages.

- ✓ At the beginning of a project, project managers have incomplete information concerning the main points of the project.
- ✓ Their info base step by step improves because the project progresses through completely different phases.
- ✓ When the completion of each section, the project managers will set up every ulterior section additional accurately and with increasing levels of confidence.

3.Software scope and feasibility

- ➢ The purpose of feasibility study is not to solve the problem, but to determine whether the problem is worth solving.
- ➢ The feasibility study concentrates on the following area.
- ➢ Operational Feasibility
- ➢ Technical Feasibility
- ➢ Economic Feasibility
- ➢ Operational Feasibility
- ➢ Operational feasibility study tests the operational scope of the software to be developed. The proposed software must have high operational feasibility. The usability will be high.

Technical Feasibility

- ➢ The technical feasibility study compares the level of technology available in the software development firm and the level of

technology required for the development of the product.

- Here the level of technology consists of the programming language, the hardware resources, Other software tools etc.

Economic Feasibility

- The economic feasibility study evaluate the cost of the software development against the ultimate income or benefits gets from the developed system.
- There must be scopes for profit after the successful Completion of the project.
- Planning provides a road map for the software development process.
- Objective of Software Project Planning
- Software Scope
- Resources
- Objective of Software Project Planning
- The objective of software project planning is to provide a frame work that enables the manager to make reasonable estimates of resources, cost and schedule.
- Software Scope
- The first activity in software project planning is the determination of software scope.
- A software project scope must be unambiguous and understandable at the management and technical levels.
- The software scope means the actual operation that is going to carried out by the software and its plus points and limitations.

Resources

- The second task of software planning is the estimation of resources required.
- Each resource is specified with the following characteristic. Resource descriptions, details of availability, when it is required, how long it is required.
- Human Resource
- Hardware Resource
- Software Resource
- People are the primary software development resource. The planner evaluate the scope and select the appropriate people for appropriate positions.

4.Resources

- Resources are people, equipment, places, money or anything else a project needs to be executed.
- As a result, resources must be allocated for each activity on the to-do list.
- Before you can assign resources to the project, however, you need to know their availability.
- Some resources need to be scheduled in advance and may only be available at certain times or times – for example, a meeting room or a rented office.
- It is therefore essential to know this before you can finish programming a project.

Resource estimation

The objective of the resource estimate is to allocate the necessary resources to each activity on the list.

There are five tools and techniques for estimating activity resources:

The judgement of experts:

this means involving experts who have already performed this type of work before and obtaining opinions on what resources are needed.

Alternative analysis:

This means considering different options on how to allocate resources.

This includes changing the number of resources and the type of resources used.

Many times, there is more than one way to perform a task, and alternative analysis helps you decide between different possibilities.

Published estimation data:

something that project managers in many industries use to understand how many resources they need for a specific project. These are based on articles, research and studies that collect, analyze and publish data from other people and organizations' projects.

Project management software: these often feature functions designed to help project managers estimate resource needs and constraints and find the best combination for the project in question.

The bottom-up estimate: this means splitting complex tasks into simpler tasks and processing the resources needed for each small step.

The need or cost of the resources of the individual tasks is then

added together to obtain a total estimate.

The smaller and more detailed the task, the greater the accuracy of this technique

Estimation of activities' duration

- Once you have finished estimating resources per activity, you have everything you need to understand how long it will take to complete each activity.
- Estimating the duration of a task means starting with information about that specific task and then working with the project team to develop a time estimate.
- Most of the time you will start with a rough estimate and then refine it.
- When you talk about estimating project time, you may have already heard of effort.
- If you're interested in learning more about it, you can check this article about effort and duration.
- Here are the five tools and techniques to create more accurate durability estimates:
- The evaluation of the experts that will come from the members of the project team who are familiar with the work that needs to be done.
- The equivalent estimate, i.e. when looking at similar activities from previous projects and how much time they took.
- Parametric Estimation, i.e. linking the project data into a formula that provides an estimation.
- The three-point estimate, i.e. when three numbers come up: a

realistic estimate that is more likely to occur, an optimistic

estimate that represents the best scenario and a pessimistic estimate that represents the worst scenario.

- The final estimate is the weighted average of the three.
- The Back-up Analysis, i.e. adding extra time to the program (called emergency reserve or buffer) to take account of additional risk.
- Activity duration estimates are a quantitative measure usually expressed in hours, weeks, days or months.
- Another thing to keep in mind when estimating activity duration is to determine the effort required.
- Duration is the amount of time an activity takes, while effort is the total number of people-hours required.
- If, for example, two people work a total of 6 hours (3 hours one and 3 hours the other) to complete an activity, the duration is six hours. However, if these two people worked the whole time (simultaneously, for 6 hours), the duration would be 12 hours.
- Project planning and critical roadmap
- The project program must be approved and signed by the stakeholders and functional managers.
- This ensures that everyone is familiar with the program, including dates and resource commitments.
- In addition, (written) confirmation will be required that resources will be available as indicated in the planning.
- Once approved, the program will become the baseline for the rest of the project.

- The progress of the project and the completion of activities will be monitored compared to project planning to determine if the project is running as planned.
- A delay in any of the activities in the critical roadmap will delay the entire project.
- project resources
- Resource equalization
- Resource equalization is used to examine and resolve the unequal use of resources, usually related to people or equipment, over time.

- During the execution of project planning, the project manager will attempt to plan certain activities simultaneously.

- As the project progresses, however, there are situations where more resources – such as equipment or people – may be needed than are available and planned.

- The project manager will attempt to schedule certain tasks at the same time as the project is progressing.

- When using project software, resource equalization can take place automatically, allowing the software to calculate delays and automatically update tasks.

- The project manager offers several tools for the development of good quantitative information, based on numbers and measurements, such as project schedules, financial and budget reports, risk analysis and objective monitoring.

- This quantitative information is essential to understand the current status and trends of a project.
- Likewise important is the development of qualitative information, such as judgement made by team members.
- In conclusion, regardless of project size or budget, estimating activities can be a challenging task.
- To create a feasible budget, the project manager needs to know their team, results, activities, and processes in detail.
- In addition, he or she should feel comfortable asking the correct questions to stakeholders.

5. Software Project Estimation

- Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.
- Estimation determines how much money, effort, resources, and time it will take to build a specific system or product.

Estimation is based on –

Past Data/Past Experience

Available Documents/Knowledge

Assumptions

Identified Risks

The four basic steps in Software Project Estimation are –

- Estimate the size of the development product.
- Estimate the effort in person-months or person-hours.
- Estimate the schedule in calendar months.
- Estimate the project cost in agreed currency.

6. Decomposition Techniques

- General Project Estimation Approach
- The Project Estimation Approach that is widely used is Decomposition Technique.
- Decomposition techniques take a divide and conquer approach. Size, Effort and Cost estimation are performed in a stepwise manner by breaking down a Project into major Functions or related Software Engineering Activities.
- Step 1 – Understand the scope of the software to be built.
- Step 2 – Generate an estimate of the software size.
- Start with the statement of scope.
- Decompose the software into functions that can each be estimated individually.
- Calculate the size of each function.
- Derive effort and cost estimates by applying the size values to your baseline productivity metrics.

- Combine function estimates to produce an overall estimate for the entire project.
- Step 3 – Generate an estimate of the effort and cost. You can arrive at the effort and cost estimates by breaking down a project into related software engineering activities
- Identify the sequence of activities that need to be performed for the project to be completed.
- Divide activities into tasks that can be measured.
- Estimate the effort (in person hours/days) required to complete each task.
- Combine effort estimates of tasks of activity to produce an estimate for the activity.
- Obtain cost units (i.e., cost/unit effort) for each activity from the database.
- Compute the total effort and cost for each activity.
- Combine effort and cost estimates for each activity to produce an overall effort and cost estimate for the entire project.

- Step 4 – Reconcile estimates: Compare the resulting values from Step 3 to those obtained from Step 2. If both sets of estimates agree, then your numbers are highly reliable. Otherwise, if widely divergent estimates occur conduct further investigation concerning whether –
- The scope of the project is not adequately understood or has been misinterpreted.

- The function and/or activity breakdown is not accurate.

- Historical data used for the estimation techniques is inappropriate for the application, or obsolete, or has been misapplied.
- Step 5 – Determine the cause of divergence and then reconcile the estimates.

7.Empricial estimation model

- Cost Estimation Models in Software Engineering
- Cost estimation simply means a technique that is used to find out the cost estimates.
- The cost estimate is the financial spend that is done on the efforts to develop and test software in Software Engineering.
- Cost estimation models are some mathematical algorithms or parametric equations that are used to estimate the cost of a product or a project.
- Various techniques or models are available for cost estimation, also known as Cost Estimation Models as shown below :

Empirical Estimation Technique –

- ✓ Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step.
- ✓ These techniques are usually based on the data that is collected previously from a project and also based on some guesses, prior experience with the development of similar types of projects, and assumptions.

- ✓ It uses the size of the software to estimate the effort.

- ✓ In this technique, an educated guess of project parameters is made.
- ✓ Hence, these models are based on common sense. However, as there are many activities involved in empirical estimation techniques, this technique is formalized.
- ✓ For example Delphi technique and Expert Judgement technique.

8.Estimation for object oriented projects

- ➢ The Object-Oriented Project Size Estimation (Oopsize) technique uses the initial estimates of B1 and B2 to predict how much time is required to design, code and test an object.
- ➢ The objects can be described in a Rational Rose class model, for instance.
- ➢ In the class model, the analyst defines object names, object attributes, object methods and parameters to object methods. Oopsize operates against the information in the Class model, calculates the Point values for each object in the model, and plugs the Point values into the predictive equation to determine an estimated amount of time for each object's creation.
- ➢ Determining the Point Values for a Set of Objects
- ➢ Point calculations are best illustrated by example. Consider the set of object definitions written in Java.

Point values for the objects are formulated as follows:

a) For MyObject1

1. Points for MyObject1 are set equal to 0.

2. The name MyObject1 is parsed. MyObject1 contains two unique tokens: My and Object1. Therefore, Points = Points + 2, or 2 total points.

3. MyObject1 has 3 attributes. Each attribute contains 1 unique token. Therefore, Points = Points + 3, or 5. Each Attribute has a type (i.e., String). Therefore, Points = Points + 3, or 8.

4. MyObject1 contains a method: Method11. Method11 contains 1 unique token. Therefore, Points = Points + 1, or 9.

5. Method11 has two arguments: Parm11 and Parm12. Therefore, Points = Points + 2, or 11. Each argument has a type. Therefore, Points = Points + 2, or 13.

6. MyObject1 contains a method: Method12. Method12 contains 1 unique token. Therefore, Points = Points + 1 or 14. Note that no points are awarded for the return value of Method12.

b) For MyObject2

1. Points for MyObject2 are set equal to 0.

2. The name MyObject2 is parsed. MyObject2 contains 1 unique token: Object2. ("My" was used previously in MyObject1). Therefore, Points = Points + 1, or 1.

3. MyObject2 contains Attribute21. It has a type (i.e. int). Therefore, Points = Points + 2, or 3.

4. MyObject2 contains Method21. Method21 contains 1 unique token. Therefore, Points = Points + 1, or 4.

5. Method21 has 1 argument: Arg1 has a type (i.e. float). Therefore, Points = Points + 2, or 6.

- Total points for MyObject1 and MyObject2 are 14 and 6, respectively.
- Then, substituting the Points values into the predictive equation on page 34, the number of days to design, code and test
- MyObject1 is 5.15 and the number of days to produce MyObject2 is 2.20.
- The total number of days for the entire project is 7.35

UNIT IV

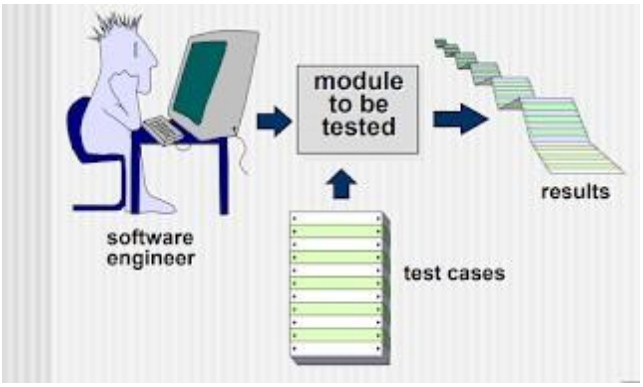
1. Testing Strategies for Conventional software
2. Validation Testing
3. System Testing
4. Testing Tactics –Software Testing Fundamentals
5. White-box Testing
6. Basis Path Testing
7. Control Structure Testing
8. Black-box Testing

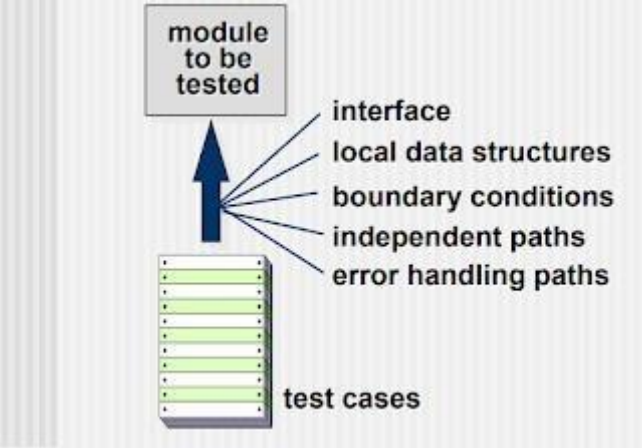
TEST STRATEGIES FOR CONVENTIONAL SOFTWARE

- There are many strategies that can be used to test software.
- At one extreme, you can wait until the system is fully constructed and then conduct tests on the overall system in hopes of finding errors.
- This approach simply does not work. It will result in buggy software.
- At the other extreme, you could conduct tests on a daily basis, whenever any part of the system is constructed.
- This approach, although less appealing to many, can be very effective.
- A testing strategy that is chosen by most software teams falls between the two extremes**.**
- It takes an incremental view of testing,
- Beginning with the testing of individual program units,
- Moving to tests designed to facilitate the integration of the units,
- Culminating with tests that exercise the constructed system.

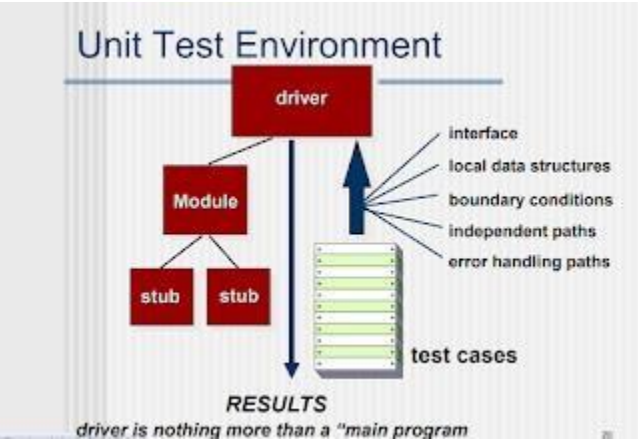
Unit Test :

- ✓ **Unit testing** focuses verification effort on the smallest unit of software design—the software component or module.
- ✓ The unit test focuses on the internal processing logic and data structures within the boundaries of a component.
- ✓ This type of testing can be conducted in parallel for multiple components.





- ✓ Unit tests are illustrated schematically in previous Figure.
- ✓ **The module interface** is tested to ensure that information properly flows into and out of the program unit under test.
- ✓ **Local data structures** are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- ✓ All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once.
- ✓ **Boundary conditions** are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
- ✓ **Finally, all error-handling paths are tested**
- ✓ Good design anticipates error conditions and establishes error-handling paths to reroute or cleanly terminate processing when an error does occur.
- ✓ *Unit testing is simplified when a component with high cohesion is designed.*



Integration Testing :

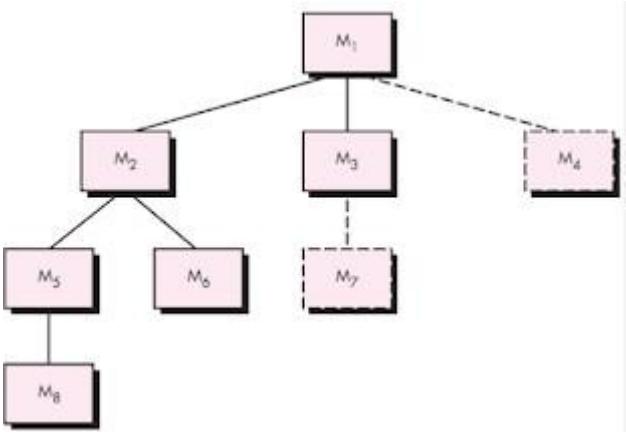
Different Integration Testing Strategies

- Top-down testing
- Bottom-up testing
- Regression Testing
- Smoke Testing

Top-down testing

Top-down integration testing is an incremental approach to construction of the software architecture.

- Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program).
- Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a *depth-first or breadth-first manner*.



Integration Testing

The integration process is performed in a series of five steps

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.

2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.

3. Tests are conducted as each component is integrated.

4. On completion of each set of tests, another stub is replaced with the real component.

5. Regression testing (discussed later in this section) may be conducted to ensure that new errors have not been introduced.

The process continues from step 2 until the entire program structure is built.

.

VALIDATION TESTING

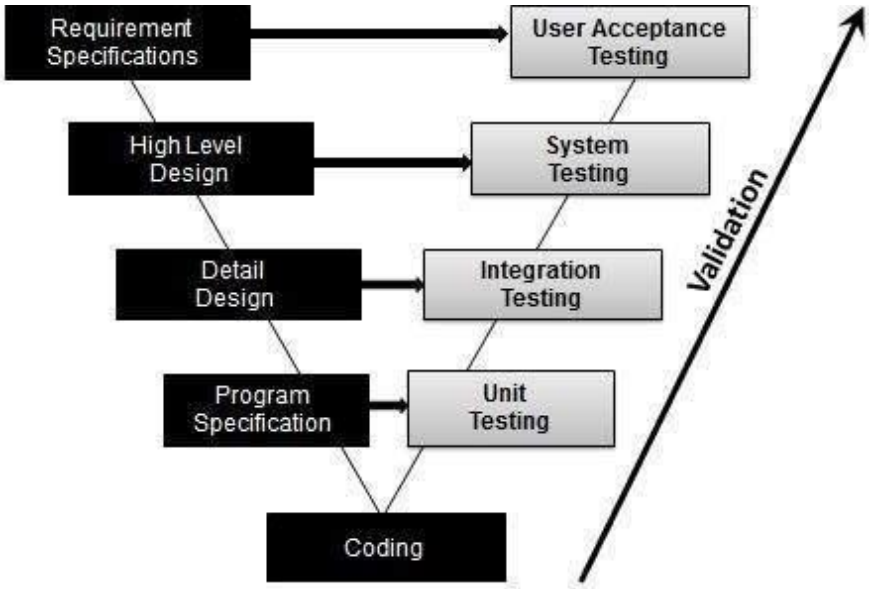
The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

Validation Testing - Workflow:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



Verification:
Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.

Verification is **Static Testing**.
Activities involved in verification:

1. Inspections
2. Reviews
3. Walkthroughs
4. Desk-checking

Verification	Validation
The process just checks the design, code and program.	It should evaluate the entire product including the code.
Reviews, walkthroughs, inspections, and desk- checking involved.	Functional and non functional methods of testing are involved . In depth check of the product is done.
It checks the software with specification.	It checks if the software meets the user needs.

ALPHA AND BETA TESTING

- Alpha Testing is performed by the Testers within the organization whereas Beta Testing is performed by the end users.
- Alpha Testing is performed at Developer's site whereas Beta Testing is performed at Client's location.
- Reliability and Security testing are not performed in-depth in Alpha Testing while Reliability, Security and Robustness are checked during Beta Testing.
- Alpha Testing involves both Whitebox and Blackbox testing whereas Beta Testing mainly involves Blackbox testing.
- Alpha Testing requires testing environment while Beta Testing doesn't require testing environment.
- Alpha Testing requires long execution cycle whereas Beta Testing requires only few weeks of execution.
- Critical issues and bugs are addressed and fixed immediately in Alpha Testing whereas issues and bugs are collected from the end users and further implemented in Beta Testing

SYSTEM TESTING

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.
In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested.

System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer. It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS).

System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing.

System Testing is a black-box testing. It is actually a series of different tests whose primary purpose is to fully exercise the computer based system.

System Testing Process:
System Testing is performed in the following steps:

- **Test Environment Setup:**
  Create testing environment for the better quality testing.
- **Create Test Case:**
  Generate test case for the testing process.
- **Create Test Data:**
  Generate the data that is to be tested.
- **Execute Test Case:**
  After the generation of the test case and the test data, test cases are executed.
- **Defect Reporting:**
  Defects in the system are detected.
- **Regression Testing:**
  It is carried out to test the side effects of the testing process.
- **Log Defects:**
  Defects are fixed in this step.
- **Retest:**
  If the test is not successful then again test is performed.

Types of System testing:

Recovery Testing

Recovery Testing is software testing technique which verifies software's ability to recover from failures like software/hardware crashes, network failures etc. The purpose of Recovery Testing is to determine whether software operations can be continued after disaster or integrity loss. Recovery testing involves reverting back software to the point where integrity was known and reprocessing transactions to the failure point.

Security Testing

SECURITY TESTING is a type of Software Testing that uncovers vulnerabilities, threats, risks in a software application and prevents malicious attacks from intruders. The purpose of Security Tests is to identify all possible loopholes and weaknesses of the software system which might result in a loss of information, revenue, repute at the hands of the employees or outsiders of the Organization.

Stress Testing

Stress testing is a software testing activity that determines the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for "mission critical" software, but is used for all types of software. A most prominent use of stress testing is to determine the limit, at which the system or software or hardware breaks**.**

Performance Testing

Performance Testing ensures software applications to perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity and stability under a particular workload.
Performance Testing is the process of analyzing the quality and capability of a product. It is a testing method performed to determine the system performance in terms of speed, reliability and stability under varying workload. Performance testing is also known as ***Perf Testing***.

Performance Testing Attributes:

- **Speed:**
  It determines whether the software product responds rapidly.
- **Scalability:**
  It determines amount of load the software product can handle at a time.
- **Stability:**
  It determines whether the software product is stable in case of varying workloads.
- **Reliability:**
  It determines whether the software product is secure or not.

Objective of Performance Testing:

1. The objective of performance testing is to eliminate performance congestion.
2. It uncovers what is needed to be improved before the product is launched in market.
3. The objective of performance testing is to make software rapid.
4. The objective of performance testing is to make software stable and reliable.

Deployment Testing

Testing a software project before and after deploying it on production is not that difficult. But too often, major bugs appear on production server after the deployment process. To avoid situations in which your production environment is threatened by these bugs, you should use a streamlined deployment and testing flow.Generally speaking, this flow consists of three phases: pre-deploy, deploy and post-deploy.

SOFTWARE TESTING FUNDAMENTALS

- **OPERABILITY**

In the context of a **software** systems, **Operability** is a measure of how well a **software** system works when operating. We say that a **software** system with good **operability** works well and is **operable ..... Software** with a high level of **operability** is easy to deploy, test, and interrogate in the Production environment.

- **OBSERVABILITY**

Observability: **Software tests** examine the outputs produced by the **software** for particular inputs. ....This implies that **software** is more testable when the **tester** has the ability to easily control the **software** in order to provide the **test** inputs.

- **CONTROLABILITY**

Controllability: As just mentioned, **software testing** involves examining outputs for given inputs. This means that the more easily we can provide inputs to the **software**, the more easily we can **test** the **software**.

- **DECOMPOSABILITY**

When software can be decomposed into independent modules, these modules can be tested individually. When an error occurs in an individual module, the error is less likely to require changes to be made in other modules, or for the tester to even examine multiple modules.

- **SIMPLICITY**

  Clearly, the simpler the software, the fewer errors it will have, and the easier it will be to test. There are three forms of simplicity that software can demonstrate: **functional simplicity**, in which the software does no more than is needed of it; **structural simplicity**, in which the software is decomposed into small, simple units; and **code simplicity**, in which the coding standards used be the software team allows for the easy understanding of the code.

- **STABILITY**

  If changes need to be made to the software, then testing becomes easier if these changes are always contained within independent modules (via, for instance, decomposability), meaning that the code that needs to be tested remains small.

- **UNDERSTANDABILITY**

  Clearly, the more the testers understand the software, the easier it is to test. Much of this relates to good software design, but also to the engineering culture of the developers: communication between designers, developers and testers whenever changes occur in the software is important, as is the ability for the testers and developers to easily access good technical documentation related to the software (such as APIs for the libraries being used and the software itself).

WHITE -BOX TESTING

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing.

It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user type perspective. On the other hand, Whitebox testing is based on the inner workings of an application and revolves around internal testing.

White box testing involves the testing of the software code for the following:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code

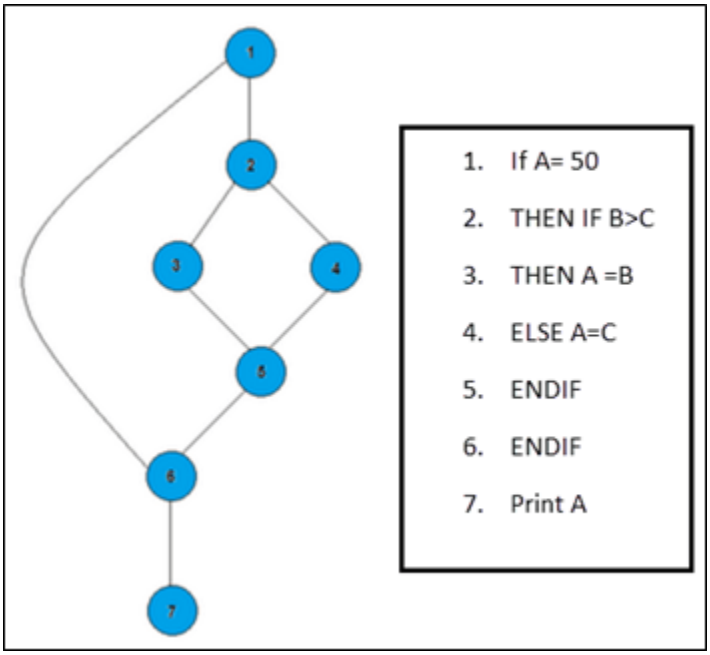
- Expected output
- The functionality of conditional loops
- Testing of each statement, object, and function on an individual basis
- Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path. It helps to determine all faults lying within a piece of code. This method is designed to execute all or selected path through a computer program.
- Any software program includes, multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming. In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

BASIS PATH TESTING

- **Basis Path Testing** in software engineering is a White Box Testing method in which test cases are defined based on flows or logical paths that can be taken through the program. The objective of basis path testing is to define the number of independent paths, so the number of test cases needed can be defined explicitly to maximize test coverage.

In software engineering, Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases. It is a hybrid method of branch testing and path testing methods.

Here we will take a simple example, to get a better idea what is basis path testing include



In the above example, we can see there are few conditional statements that is executed depending on what condition it suffice. Here there are 3 paths or condition that need to be tested to get the output,

- **Path 1**: 1,2,3,5,6, 7
- **Path 2**: 1,2,4,5,6, 7
- **Path 3**: 1, 6, 7

Steps for Basis Path testing

The basic steps involved in basis path testing include

- Draw a control graph (to determine different program paths)
- Calculate Cyclomatic complexity (metrics to determine the number of independent paths)
- Find a basis set of paths
- Generate test cases to exercise each path

Advantages of Basic Path Testing

- It helps to reduce the redundant tests
- It focuses attention on program logic
- It helps facilitates analytical versus arbitrary case design
- Test cases which exercise basis set will execute every statement in a program at least once

CONTROL STRUCTURE TESTING

Control structure testing is used to increase the coverage area by testing various control structures present in the program. The different types of testing performed under control structure testing are as follows-

1. Condition Testing
2. Data Flow Testing
3. Loop Testing

1. Condition Testing :
Condition testing is a test cased design method, which ensures that the logical condition and decision statements are free from errors. The errors present in logical conditions can be incorrect boolean operators, missing parenthesis in a booleans expression, error in relational operators, arithmetic expressions, and so on.
The common types of logical conditions that are tested using condition testing are-

1. A relation expression, like E1 op E2 where ‘E1’ and ‘E2’ are arithmetic expressions and ‘OP’ is an operator.

2. A simple condition like any relational expression preceded by a NOT (~) operator. For example, (~E1) where 'E1' is an arithmetic expression and 'a' denotes NOT operator.
3. A compound condition consists of two or more simple conditions, Boolean operator, and parenthesis.
   For example, (E1 & E2)|(E2 & E3) where E1, E2, E3 denote arithmetic expression and '&' and '|' denote AND or OR operators.

4. A Boolean expression consists of operands and a Boolean operator like 'AND', OR, NOT.
   For example, 'A|B' is a Boolean expression where 'A' and 'B' denote operands and | denotes OR operator.

2. Data Flow Testing :

The data flow test method chooses the test path of a program based on the locations of the definitions and uses all the variables in the program.
The data flow test approach is depicted as follows suppose each statement in a program is assigned a unique statement number and that theme function cannot modify its parameters or global variables.
For example, with S as its statement number.

```
DEF (S) = {X | Statement S has a definition of X}

USE (S) = {X | Statement S has a use of X}
```

If statement S is an if loop statement, them its DEF set is empty and its USE set depends on the state of statement S. The definition of the variable X at statement S is called the line of statement S' if the statement is any way from S to statement S' then there is no other definition of X.

A definition use (DU) chain of variable X has the form [X, S, S'], where S and S' denote statement numbers, X is in DEF(S) and USE(S'), and the definition of X in statement S is line at statement S'.

A simple data flow test approach requires that each DU chain be covered at least once. This approach is known as the DU test approach. The DU testing does not ensure coverage of all branches of a program.

However, a branch is not guaranteed to be covered by DU testing only in rar cases such as then in which the other construct does not have any certainty of any variable in its later part and the other part is not present. Data flow testing strategies are appropriate for choosing test paths of a program containing nested if and loop statements.

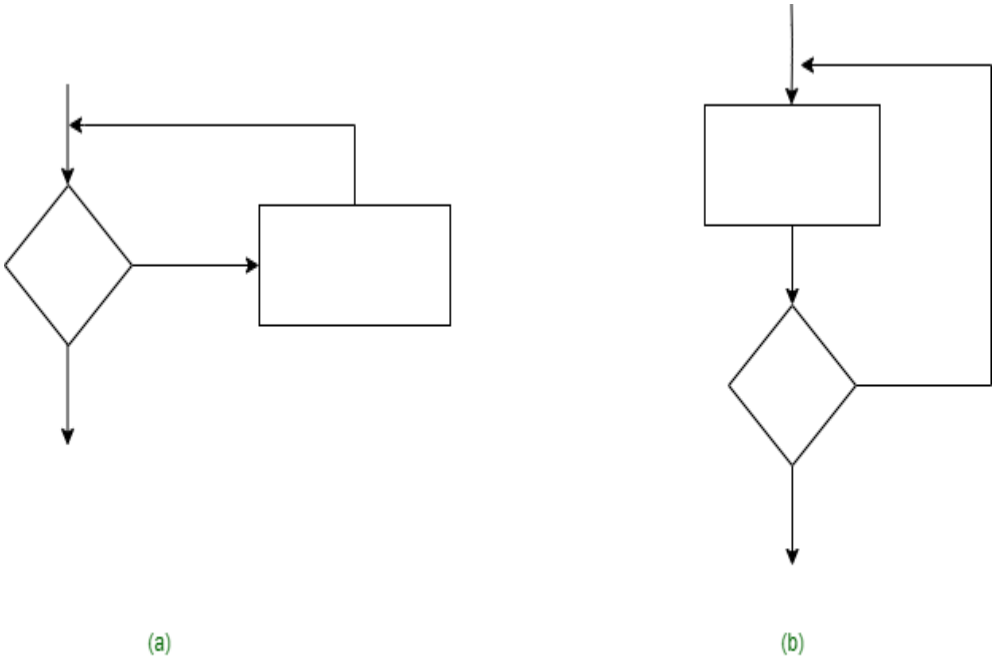
3. Loop Testing :

Loop testing is actually a white box testing technique. It specifically focuses on the validity of

loop construction.

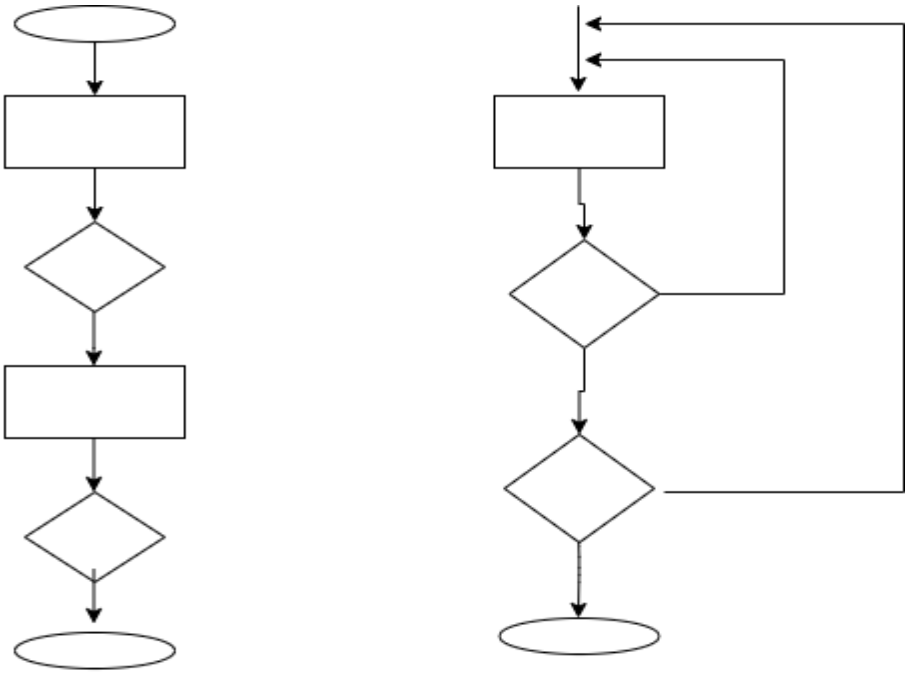
Following are the types of loops.

1. **Simple Loop –** The following set of test can be applied to simple loops, where the maximum allowable number through the loop is n.
    1. Skip the entire loop.
    2. Traverse the loop only once.
    3. Traverse the loop two times.
    4. Make p passes through the loop where p<n.
    5. Traverse the loop n-1, n, n+1 times.



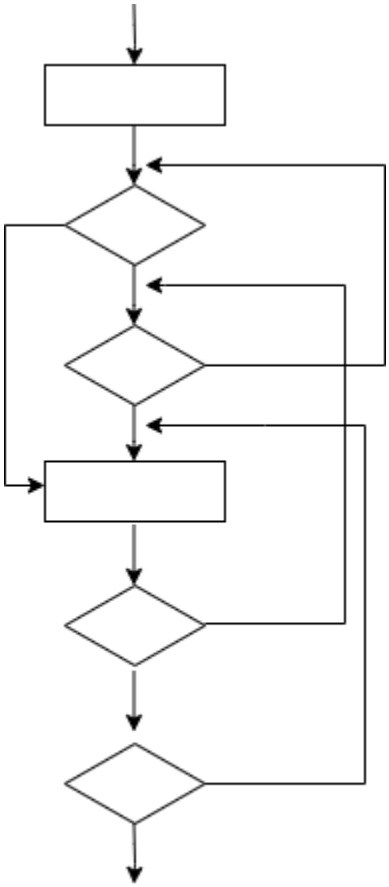
SIMPLE LOOPS

2. **Concatenated Loops –** If loops are not dependent on each other, contact loops can be tested using the approach used in simple loops. if the loops are interdependent, the steps are followed in nested loops.



Concatenated Loops

3. **Nested Loops –** Loops within loops are called as nested loops. when testing nested loops, the number of tested increases as level nesting increases.
   The following steps for testing nested loops are as follows-
   1. Start with inner loop. set all other loops to minimum values.
   2. Conduct simple loop testing on inner loop.
   3. Work outwards.
   4. Continue until all loops tested.
4. **Unstructured loops –** This type of loops should be redesigned, whenever possible, to reflect the use of unstructured the structured programming constructs.



Unstructured Loops

BLACK BOX TESTING

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

Types of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones -

- **Functional testing** - This black box testing type is related to the functional requirements of a system; it is done by software testers.
- **Non-functional testing** - This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** - Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Black Box Testing	White Box Testing
the main focus of black box testing is on the validation of your functional requirements.	White Box Testing (Unit Testing) validates internal structure and working of your software code
Black box testing gives abstraction from code and focuses on testing effort on the software system behavior.	To conduct White Box Testing, knowledge of underlying programming language is essential. Current day software systems use a variety of programming languages and technologies and its not possible to know all of them.
Black box testing facilitates testing communication amongst modules	White box testing does not facilitate testing communication amongst modules

Unit V

METRICS FOR PROCESS AND PROJECTS

1. Metrics in the process and project domain
2. Software measurements
3. Metrics for Software quality
4. Integrating Metrics within the Software Process
5. Engineering of Component Based System(CBSE)
6. The CBSE process
7. Domain engineering
8. CBD
9. Classifying and Retrieving Components
10. Economics of CBSE

1. Metrics in the process and project domain

Project managers have a wide variety of metrics to choose from.

We can classify the most commonly used metrics into the following groups:

Process Metrics

These are metrics that pertain to Process Quality. They are used to measure the efficiency and effectiveness of various processes.

Project Metrics

These are metrics that relate to Project Quality. They are used to quantify defects, cost, schedule, productivity and estimation of various project resources and deliverables.

Product Metrics

These are metrics that pertain to Product Quality. They are used to measure cost, quality, and the product's time-to-market.

Organizational Metrics

These metrics measure the impact of organizational economics, employee satisfaction, communication, and organizational growth factors of the project.

Software Development Metrics Examples

- These metrics enable management to understand the quality of the software, the productivity of the development team, code complexity, customer satisfaction, agile process, and operational metrics.

- We'll now take a closer look at the various types of the two most important categories of metrics – Project Metrics, and Process Metrics.
- Schedule Variance: Any difference between the scheduled completion of an activity and the actual completion is known as Schedule Variance.
- Schedule variance = ((Actual calendar days – Planned calendar days) + Start variance)/ Planned calendar days x 100.
- Effort Variance: Difference between the planned outlined effort and the effort required to actually undertake the task is called Effort variance.
- Effort variance = (Actual Effort – Planned Effort)/ Planned Effort x 100.
- Size Variance: Difference between the estimated size of the project and the actual size of the project (normally in KLOC or FP).
- Size variance = (Actual size – Estimated size)/ Estimated size x 100.
- Requirement Stability Index: Provides visibility to the magnitude and impact of requirements changes.
- RSI = 1- ((Number of changed + Number of deleted + Number of added) / Total number of initial requirements) x100.
- Productivity (Project): Is a measure of output from a related process for a unit of input.

- Project Productivity = Actual Project Size / Actual effort expended in the project.

- Productivity (for test case preparation) = Actual number of test cases/ Actual effort expended in test case preparation.
- Productivity (for test case execution) = Actual number of test cases / actual effort expended in testing.
- Productivity (defect detection) = Actual number of defects (review + testing) / actual effort spent on (review + testing).
- Productivity (defect fixation) = actual no of defects fixed/ actual effort spent on defect fixation.
- Schedule variance for a phase: The deviation between planned and actual schedules for the phases within a project
- Schedule variance for a phase = (Actual Calendar days for a phase – Planned calendar days for a phase + Start variance for a phase)/ (Planned calendar days for a phase) x 100
- Effort variance for a phase: The deviation between a planned and actual effort for various phases within the project.
- Effort variance for a phase = (Actual effort for a phase – a planned effort for a phase)/ (planned effort for a phase) x 100.
- Process Metrics:
- Cost of quality: It is a measure of the performance of quality initiatives in an organization. It's expressed in monetary terms.

- Cost of quality = (review + testing + verification review + verification testing + QA + configuration management + measurement + training + rework review + rework testing)/ total effort x 100.

- Cost of poor quality: It is the cost of implementing imperfect processes and products.

- Cost of poor quality = rework effort/ total effort x 100.

- Defect density: It is the number of defects detected in the software during development divided by the size of the software (typically in KLOC or FP)

- Defect density for a project = Total number of defects/ project size in KLOC or FP

- Review efficiency: defined as the efficiency in harnessing/ detecting review defects in the verification stage.

- Review efficiency = (number of defects caught in review)/ total number of defects caught) x 100.

- Testing Efficiency: Testing efficiency = 1 – ((defects found in acceptance)/ total number of testing defects) x 100.

- Defect removal efficiency: Quantifies the efficiency with which defects were detected and prevented from reaching the customer.

- Defect removal efficiency = (1 – (total defects caught by customer/ total number of defects)) x 100.

- Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

- Check out our course on Introduction to PMP Certification Training.

2. Software Measurement

- A measurement is an manifestation of the size, quantity, amount or dimension of a particular attributes of a product or process.
- Software measurement is a titrate impute of a characteristic of a software product or the software process. It is an authority within software engineering.

- Software measurement process is defined and governed by ISO Standard.

Need of Software Measurement:

Software is measured to:

- ❖ Create the quality of the current product or process.
- ❖ Anticipate future qualities of the product or process.
- ❖ Enhance the quality of a product or process.
- ❖ Regulate the state of the project in relation to budget and schedule.
- ❖ Classification of Software Measurement:

There are 2 types of software measurement:

Direct Measurement:

- In direct measurement the product, process or thing is measured directly using standard scale.
- Indirect Measurement:
- In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference.

Metrics:

A metrics is a measurement of the level that any impute belongs to a system product or process.

There are 4 functions related to software metrics:

Planning

Organizing

Controlling

Improving

Characteristics of software Metrics:

Quantitative:

Metrics must possess quantitative nature.It means metrics can be expressed in values.

Understandable:

Metric computation should be easily understood ,the method of computing metric should be clearly defined.

Applicability:

Metrics should be applicable in the initial phases of development of the software.

Repeatable:

The metric values should be same when measured repeatedly and consistent in nature.

Economical:

Computation of metric should be economical.

Language Independent:

Metrics should not depend on any programming language.

Classification of Software Metrics:

There are 2 types of software metrics:

Product Metrics:

Product metrics are used to evaluate the state of the product, tracing risks and undercovering prospective problem areas. The ability of team to control quality is evaluated.

Process Metrics:

Process metrics pay particular attention on enhancing the long term process of the team or organisation.

Project Metrics:

Project matrix is describes the project characteristic and execution process.

Number of software developer

Staffing pattern over the life cycle of software

Cost and schedule

Productivity

3. Metrics for Software quality

- Software quality metrics are a subset of software metrics that focus on the quality aspects of the product, process, and project.
- These are more closely associated with process and product metrics than with project metrics.

- Software quality metrics can be further divided into three categories –
- Product quality metrics
- In-process quality metrics
- Maintenance quality metrics
- Product Quality Metrics

This metrics include the following –

- Mean Time to Failure
- Defect Density
- Customer Problems
- Customer Satisfaction
- Mean Time to Failure
- It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

Defect Density

- It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit.
- This metric is used in many commercial software systems.
- Customer Problems

- It measures the problems that customers encounter when using the product.
- It contains the customer's perspective towards the problem space of the software, which includes the non-defect oriented problems together with the defect problems.
- The problems metric is usually expressed in terms of Problems per User-Month (PUM)

4. Integrating Metrics Within the Software Process

Arguments for Software Metrics

- Why is it so important to measure the process of software engineering and the product (software) that it produces?
- The answer is relatively obvious.
- If we do not measure, there no real way of determining whether we are improving.
- And if we are not improving, we are lost.
- By requesting and evaluating productivity and quality measures, senior management can establish meaningful goals for improvement of the software engineering process.
- If the process through which it is developed can be improved, a direct impact on the bottom line can result.
- But to establish goals for improvement, the current status of software development must be understood.
- Hence, measurement is used to establish a process baseline from which improvements can be assessed.

- The day-to-day rigors of software project work leave little time for strategic thinking.
- Software project managers are concerned with more mundane (but equally important)
- issues: developing meaningful project estimates, producing higher-quality systems, getting product out the door on time.
- By using measurement to establish a project baseline, each of these issues becomes more manageable.
- We have already noted that the baseline serves as a basis for estimation.
- Additionally, the collection of quality metrics enables an organization to "tune" its software process to remove the "vital few" causes of defects that have the greatest impact on software development.
- At the project and technical levels (in the trenches), software metrics provide immediate benefit.
- As the software design is completed, most developers would be anxious to obtain answers to the questions such as
    - Which user requirements are most likely to change?
    - Which components in this system are most error prone?
    - How much testing should be planned for each component?
    - How many errors (of specific types) can I expect when testing commences?

- Establishing a Baseline
- By establishing a metrics baseline, benefits can be obtained at the process, project, and product (technical) levels. Yet the information that is collected need not be fundamentally different.
- The same metrics can serve many masters.
- The metrics baseline consists of data collected from past software development projects To be an effective aid in process improvement and/or cost and effort estimation,
- baseline data must have the following attributes:
- data must be reasonably accurate—"guestimates" about past projects are to be avoided;
- data should be collected for as many projects as possible;
- measures must be consistent, for example, a line of code must be interpreted consistently across all projects for which data are collected;
- applications should be similar to work that is to be estimated—it makes little sense to use a baseline for batch information systems work to estimate a realtime, embedded application.
- Metrics Collection, Computation, and Evaluation
- Ideally, data needed to establish a baseline has been collected in an ongoing manner.
- Sadly, this is rarely the case. Therefore, data collection requires a historical investigation of past projects to reconstruct

required data. Once measures have been collected (unquestionably the most difficult step), metrics computation is possible.

- Depending on the breadth of measures collected, metrics can span a broad range of LOC or FP metrics as well as other quality- and project-oriented metrics.
- Finally, metrics must be evaluated and applied during estimation, technical work, project control, and process improvement.
- Metrics evaluation focuses on the underlying reasons for the results obtained and produces a set of indicators that guide the project or process.

5.Component Based Software Engineering (CBSE)

It is a process that focuses on the design and development of computer-based systems with the use of reusable software components.

CBSE Framework Activities

Framework activities of Component Based Software Engineering are as follows:-

Component Qualification:

This activity ensures that the system architecture define the requirements of the components for becoming a reusable component.Reusable components are generally identified through the traits in their interfaces.

It means "the services that are given, and the means by which customers or consumers access these services " are defined as a part of the component interface.

Component Adaptation:

This activity ensures that the architecture defines the design conditions for all component and identifying their modes of connection.

In some of the cases, existing reusable components may not be allowed to get used due to the architecture's design rules and conditions.

These components should adapt and meet the requirements of the architecture or refused and replaced by other, more suitable components.

Component Composition:

This activity ensures that the Architectural style of the system integrates the software components and form a working system.

By identifying connection and coordination mechanisms of the system, the architecture describes the composition of the end product.

Component Update:

This activity ensures the updation of reusable components. Sometimes, updates are complicated due to inclusion of third party (the organization that developed the reusable component may be outside the immediate control of the software engineering organization accessing the component currently.)

6. TheCBSE process

- CBSE processes are software processes that support component-based software engineering.
- They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components.

There are two types of CBSE processes:

- CBSE for reuse is concerned with developing components or services that will be reused in other applications.
- It usually involves generalizing existing components.
- CBSE with reuse is the process of developing new applications using existing components and services.
- CBSE for reuse focuses on component and service development. Components developed for a specific application usually have to be generalised to make them reusable.
- A component is most likely to be reusable if it associated with a stable domain abstraction (business object).
- For example, in a hospital stable domain abstractions are associated with the fundamental purpose - nurses, patients, treatments, etc.

Component reusability:

- Should reflect stable domain abstractions;
- Should hide state representation;
- Should be as independent as possible;

- Should publish exceptions through the component interface.
- There is a trade-off between reusability and usability. The more general the interface, the greater the reusability but it is then more complex and hence less usable.
- To make an existing component reusable:
- Remove application-specific methods.
- Change names to make them general.
- Add methods to broaden coverage.
- Make exception handling consistent.
- Add a configuration interface for component adaptation.
- Integrate required components to reduce dependencies.
- Existing legacy systems that fulfill a useful business function can be re-packaged as components for reuse.
- This involves writing a wrapper component that implements provides and requires interfaces then accesses the legacy system.
- Although costly, this can be much less expensive than rewriting the legacy system.

7. Domain engineering

- It's also called product line engineering, is the entire process of reusing domain knowledge in the production of new software systems.

- It is a key concept in systematic software reuse. A key idea in systematic software reuse is the domain.
- Most organizations work in only a few domains.
- They repeatedly build similar systems within a given domain with variations to meet different customer needs.
- Rather than building each new system variant from scratch, significant savings may be achieved by reusing portions of previous systems in the domain to build new ones.
- The process of identifying domains, bounding them, and discovering commonalities and variabilities among the systems in the domain is called domain analysis.
- This information is captured in models that are used in the domain implementation phase to create artifacts such as reusable components, a domain-specific language, or application generators that can be used to build new systems in the domain.
- 8.CBD
- Component-based development techniques consist of non-conventional development routines, including component evaluation, component retrieval, etc. It is important that the CBD is carried out within a middleware infrastructure that supports the process, for example, Enterprise Java Beans.

- The key goals of CBD are as follows:

- Save time and money when building large and complex systems: Developing complex software systems with the help of off-the-shelf components helps reduce software development time substantially. Function points or similar techniques can be used to verify the affordability of the existing method.

- Enhance the software quality: The component quality is the key factor behind the enhancement of software quality.

- Detect defects within the systems: The CBD strategy supports fault detection by testing the components; however, finding the source of defects is challenging in CBD.
- Some advantages of CBD include:

- Minimized delivery:
- Search in component catalogs
- Recycling of pre-fabricated components

- Improved efficiency:
- Developers concentrate on application development

- Improved quality:

- Component developers can permit additional time to ensure quality

- Minimized expenditures
- The specific routines of CBD are:

- Component development
- Component publishing
- Component lookup as well as retrieval
- Component analysis
- Component assembly

9. Classifying And Retrieving Components

- ✓ Consider a large university library.
- ✓ Tens of thousands of books, periodicals, and other information resources are available for use.
- ✓ But to access these resources, a categorization scheme must be developed.
- ✓ To navigate this large volume of information, librarians have defined a classification scheme that includes a Library of Congress classification code, keywords, author names, and other index entries. All enable the user to find the needed resource quickly and easily.

- ✓ Now, consider a large component repository.
- ✓ Tens of thousands of reusable software components reside in it. But how does a software engineer find the one she needs?
- ✓ To answer this question, another question arises: How do we describe software components in unambiguous, classifiable terms? These are difficult questions, and no definitive answer has yet been developed.
- ✓ In this section we explore current directions that will enable future software engineers to navigate reuse libraries.

Describing Reusable Components

- ➢ A reusable software component can be described in many ways, but an ideal description encompasses what Tracz has called the 3C model—concept, content, and context.
- ➢ The concept of a software component is "a description of what the component does" .
- ➢ The interface to the component is fully described and the semantics— represented within the context of pre- and postconditions—are identified.
- ➢ The concept should communicate the intent of the component.
- ➢ The content of a component describes how the concept is realized.

- In essence, the content is information that is hidden from casual users and need be known only to those who intend to modify or test the component.

- The context places a reusable software component within its domain of applicability.
- That is, by specifying conceptual, operational, and implementation features, the context enables a software engineer to find the appropriate component to meet application requirements.

- To be of use in a pragmatic setting, concept, content, and context must be translated into a concrete specification scheme.
- Dozens of papers and articles have been written about classification schemes for reusable software components .
- The methods proposed can be categorized into three major areas: library and information science methods, artificial intelligence methods, and hypertext systems.
- The vast majority of work done to date suggests the use of library science methods for component classification.

- Figure presents a taxonomy of library science indexing methods. Controlled indexing vocabularies limit the terms or syntax that can be used to classify an object (component).

Uncontrolled indexing vocabularies place no restrictions on the nature of the description. The majority of classification schemes

10.Economics of CBSE

- Component-based software engineering has an intuitive appeal.
- In theory, it should provide a software organization with advantages in quality and timeliness.
- And these should translate into cost savings. But are there hard data that support our intuition?
- To answer this question we must first understand what actually can be reused in a software engineering context and then what the costs associated with reuse really are.
- As a consequence, it is possible to develop a cost/benefit analysis for component reuse.
- Impact on Quality, Productivity, and Cost
- Considerable evidence from industry case studies indicates substantial business benefits can be derived from aggressive software reuse.
- Product quality, development productivity, and overall cost are all improved.

- Quality. In an ideal setting, a software component that is developed for reuse would be verified to be correct and would contain no defects.

- In reality, formal verification is not carried out routinely, and defects can and do occur. However, with each reuse, defects are found and eliminated, and a component's quality improves as a result. Over time, the component becomes virtually defect free.

- In a study conducted at Hewlett Packard, Lim [LIM94] reports that the defect rate for reused code is 0.9 defects per KLOC, while the rate for newly developed software is 4.1 defects per KLOC.
- For an application that was composed of 68 percent reused code, the defect rate was 2.0 defects per KLOC—a 51 percent improvement from the expected rate, had the application been developed without reuse.
- Henry and Faller report a 35 percent improvement in quality. Although anecdotal reports span a reasonably wide spectrum of quality improvement percentages, it is fair to state that reuse provides a nontrivial benefit in terms of the quality and reliability for delivered software.

- Productivity. When reusable components are applied throughout the software process, less time is spent creating the plans, models, documents, code, and data that are required to create a deliverable system.

- It follows that the same level of functionality is delivered to the customer with less input effort. Hence, productivity is improved.
- Although percentage productivity improvement reports are notoriously difficult to interpret, it appears that 30 to 50 percent reuse can result in productivity improvements in the 25–40 percent range.

- Cost. The net cost savings for reuse are estimated by projecting the cost of the project if it were developed from scratch, Cs, and then subtracting the sum of the costs associated with reuse, Cr, and the actual cost of the software as delivered,
- Cd. Cs can be determined by applying one or more of the estimation techniques The costs associated with reuse, Cr, include

  - Domain analysis and modeling.
  - Domain architecture development.
  - Increased documentation to facilitate reuse.
  - Support and enhancement of reuse components.
  - Royalties and licenses for externally acquired components.
  - Creation or acquisition and operation of a reuse repository.

- Training of personnel in design and construction for reuse.

➢ Although costs associated with domain analysis and the operation of a reuse repository can be substantial, many of the other costs noted here address issues that are part of good software engineering practice, whether or not reuse is a priority.