

## II B. TECH II SEM

Subject Name: **SOFTWARE ENGINEERING**

Regulation: **R20**

### Course Objectives:

- To help students to develop skills that will enable them to construct software of high quality – software that is reliable, and that is reasonably easy to understand, modify and maintain.
- This course introduces the concepts and methods required for the construction of large software intensive systems. It aims to develop a broad understanding of the discipline of software engineering.
- Capable of team and organizational leadership in computing project settings, and have a broad understanding of ethical application of computing-based solutions to societal and organizational problems.
- Apply their foundations in software engineering to adapt to readily changing environments using the appropriate theory, principles and processes
- Represent classes, responsibilities and states using UML notation

### UNIT-I:

**Software and Software Engineering:** The Nature of Software, The Unique Nature of WebApps, Software Engineering, Software Process, Software Engineering Practice, Software Myths.

**Process Models:** A Generic Process Model, Process Assessment and Improvement, Prescriptive Process Models, Specialized Process Models, The Unified Process, Personal and Team Process Models

### UNIT-II:

**Requirements Analysis and Specification:** Requirements Gathering and Analysis, Software Requirement Specification (SRS), Formal System Specification.

**Software Design:** Overview of the Design Process, How to Characterize of a Design? Cohesion and Coupling, Layered Arrangement of Modules, Approaches to Software Design, Developing the DFD Model of a System

### UNIT – III:

**Unified Modeling Language (UML):** Introduction to UML, why we model, Standard Diagrams: Structural Diagrams- Class diagram, Object diagram, Component diagram, Deployment diagram, Behavioural Diagrams-Use case diagram, Sequence diagram, Collaboration diagram, State chart diagram, Activity diagram.

M.Srikanth,  
M.Tech., (Ph.D)  
Asst.Prof, Dept of IT, VIT

## **UNIT – IV:**

**Coding And Testing:** Coding, Code Review, Software Documentation, Testing, Unit Testing, Black-Box Testing, White-Box Testing, Debugging, Program Analysis Tool, Integration Testing, Testing Object-Oriented Programs, System Testing, Some General Issues Associated with Testing.

## **UNIT – V:**

**Software Reliability and Quality Management:** Software Reliability, Statistical Testing, Software Quality, Software Quality Management System, ISO 9000, SEI Capability Maturity Model.

**Software Maintenance:** Software maintenance, Maintenance Process Models, Maintenance Cost, Software Configuration Management.

## **OUTCOMES:**

- Define and develop a software project from requirement gathering to implementation.
- Obtain knowledge about principles and practices of software engineering.
- Focus on the fundamentals of modelling a software project.
- Obtain knowledge about estimation and maintenance of software systems

## **TEXT BOOKS:**

1. Software Engineering a Practitioner's Approach, Roger S. Pressman, Seventh Edition McGrawHill International Edition.
2. Fundamentals of Software Engineering, Rajib Mall, Third Edition, PHI.
3. Software Engineering, Ian Sommerville, Ninth edition, Pearson education
4. The Unified Modeling Language User Guide", Grady Booch, James Rumbaugh, Ivar Jacobson, 12<sup>th</sup> Impression, 2012, PEARSON.

## **REFERENCE BOOKS:**

1. Software Engineering : A Primer, Waman S Jawadekar, Tata McGraw-Hill, 2008
2. Software Engineering, A Precise Approach, PankajJalote, Wiley India,2010.
3. Software Engineering, Principles and Practices, Deepak Jain, Oxford University Press.
4. Software Engineering1: Abstraction and modeling, Diner Bjorner, Springer International edition, 2006.

M.Srikanth,  
M.Tech., (Ph.D)  
Asst.Prof, Dept of IT, VIT

# UNIT-I

## 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.

### Why learn software engineering

Software engineering is important because specific software is needed in almost every industry, in every business, and for every function. It becomes more important as time goes on – if something breaks within your application portfolio, a quick, efficient, and effective fix needs to happen as soon as possible.

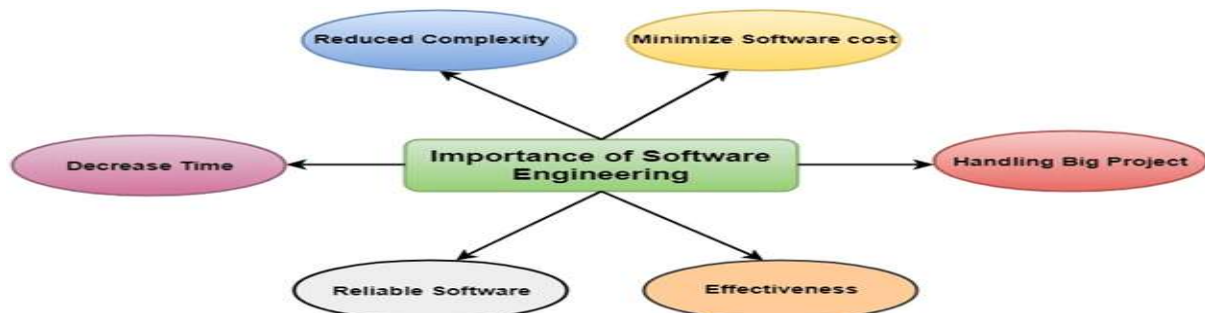
## Need of Software Engineering

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

- Huge Programming
- Adaptability
- Cost
- Dynamic Nature
- Quality Management

## Importance of Software Engineering

1. **Reduces complexity:** big software is always complicated and challenging to progress. Software engineering divides big problems into various small issues. All these small problems are solved independently to each other.
2. **To minimize software cost:** A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed.
3. **To decrease time:** So if you are making your software according to the software engineering method, then it will decrease a lot of time.
4. **Handling big projects:** projects are not done in a couple of days, and they need lots of patience, planning, and management. No one can say that he has given four months of a company to the task, and the project is still in its first stage.
5. **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
6. **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So Software becomes more effective in the act with the help of software engineering.



## THE NATURE OF SOFTWARE

- Software is among the most complex of engineered artifacts.
- Software is flexible, so is expected to conform to standards imposed by other components, Ex: hardware, external agents etc.
- Flexibility also increases the rate at which software is changed during its lifetime.
- The invisibility of software makes it harder to contextualize compared to other engineering sectors, Ex: construction industry.

**Instruction:** computer program that when executed provide desired features, function, and performance

**Data structures:** That enable the programs to effectively manipulate information

**Descriptive:** Information in both hard copy and virtual forms that describes the operation and use of the programs

## THE UNIQUE NATURE OF WEBAPPS

In the early days of the World Wide Web (1990 to 1995), websites consisted of little more than a set of linked hypertext files that presented information using text and limited graphics. Today, WebApps have evolved into sophisticated computing tools that not only provide stand-alone function to the end user, but also have been integrated with corporate databases and business applications due to the development of HTML, JAVA, xml etc.

### Attributes of WebApps:

- Network Intensiveness
- Concurrency
- Unpredictable load
- Performance
- Availability
- Data driven
- Content Sensitive
- Continuous evolution
- Immediacy
- Security
- Aesthetic

### Network intensiveness.

A WebApp resides on a network and must serve the needs of a diverse community of clients. The network may enable worldwide access and communication (i.e., the Internet) or more limited access and communication.

(e.g., a corporate Intranet Network Intensiveness)

**Concurrency:** A large number of users may access the WebApp at one time. In many cases, the patterns of usage among end users will vary greatly.

**Unpredictable load:** The number of users of the WebApp may vary by orders of magnitude from day to day. One hundred users may show up on Monday; 10,000 may use the system on Thursday.

**Performance:** If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.

**Availability:** Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a 24/7/365 basis.

**Data driven:** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end user.

In addition, WebApps are commonly used to access information that exists on databases that are not an integral part of the Web-based environment (e.g., e-commerce or financial applications).

**Content sensitive:** The quality and artistic nature of content remains an important Determinant of the quality of a WebApp.

**Continuous evolution:** Unlike conventional application software that evolves over a series of planned, chronologically spaced releases, Web applications evolve continuously. It is not unusual for some WebApps (specifically, their content) to be updated on a minute-by-minute schedule or for content to be independently computed for each request.

**Immediacy:** Although immediacy—the compelling (forceful) need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time-to-market that can be a matter of a few days or weeks.

**Security:** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end users who may access the application. In order to protect sensitive content and provide secure mode of data transmission, strong security measures must be implemented.

**Aesthetics:** An undeniable part of the appeal of a WebApp is its look and feel. When an application has been designed to market or sell products or ideas, aesthetic may have as much to do with success as technical design.

## SOFTWARE PROCESS

Software is the set of instructions in the form of programs to govern the computer system and to process the hardware components. To produce a software product the set of activities is used. This set is called a software process.

**Software Development:** In this process, designing, programming, documenting, testing, and bug fixing is done.

**Components of Software:**

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

1. Program – A computer program is a list of instructions that tell a computer what to do.
2. Documentation – Source information about the product contained in design documents, detailed code comments, etc.
3. Operating Procedures – Set of step-by-step instructions compiled by an organization to help workers carry out complex routine operations.

**There are four basic key process activities:**

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

**Software Crisis:**

1. Size and Cost – Day to day growing complexity and expectation out of software. Software is more expensive and more complex.
2. Quality– Software products must have good quality.
3. Delayed Delivery – Software takes longer than the estimated time to develop, which in turn leads to cost shooting up.

**Software Process Model:**

A software process model is an abstraction of the actual process, which is being described. It can also be defined as a simplified representation of a software process. Each model represents a process from a specific perspective. Basic software process models on which different type of software process models can be implemented:

1. A workflow Model – It is the sequential series of tasks and decisions that make up a business process.

2. The Waterfall Model – It is a sequential design process in which progress is seen as flowing steadily downwards. Phases in waterfall model:
  - (i) Requirements Specification
  - (ii) Software Design
  - (iii) Implementation
  - (iv) Testing
3. Dataflow Model – It is diagrammatic representation of the flow and exchange of information within a system.
4. Evolutionary Development Model – Following activities are considered in this method:
  - (i) Specification
  - (ii) Development
  - (iii) Validation
5. Role / Action Model – Roles of the people involved in the software process and the activities.

## **SOFTWARE ENGINEERING PRACTICE**

- Software engineering practice is a broad array of principles, concepts, methods, and tools that you must consider as software is planned and developed.
- In a generic sense, practice is a collection of concepts, principles, methods, and tools that a software engineer calls upon on a daily basis.
- Practice allows managers to manage software projects and software engineers to build computer programs
- Practice populates a software process model with the necessary technical and management how to s to get job done.
- Practice transforms a haphazard unfocused approach into something that is more organized more effective and more likely to achieve success.

### **The essence of practice:**

1. Understand the problem (communication and analysis)
2. Plan a solution (modelling and software design)
3. Carry out the plan (code generation)
4. Examine the result for accuracy (testing and quality assurance)



### Understand the problem.

- *Who has a stake in the solution to the problem?* That is, who are the stakeholders?
- *What are the unknowns?* What data, functions, features, and behavior are required to properly solve the problem?
- *Can the problem be compartmentalized?* Is it possible to represent smaller problems that may be easier to understand?
- *Can the problem be represented graphically?* Can an analysis model be created?

### Plan the solution.

- *Have you seen similar problems before?* Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, features, and behavior that are required?
- *Has a similar problem been solved?* If so, are elements of the solution reusable?
- *Can subproblems be defined?* If so, are solutions readily apparent for the subproblems?
- *Can you represent a solution in a manner that leads to effective implementation?* Can a design model be created?

### Carry out the plan.

- *Does the solution conform to the plan?* Is source code traceable to the design model?
- *Is each component part of the solution probably correct?* Has the design and code been reviewed, or better, have correctness proofs been applied to the algorithm?

### Examine the result.

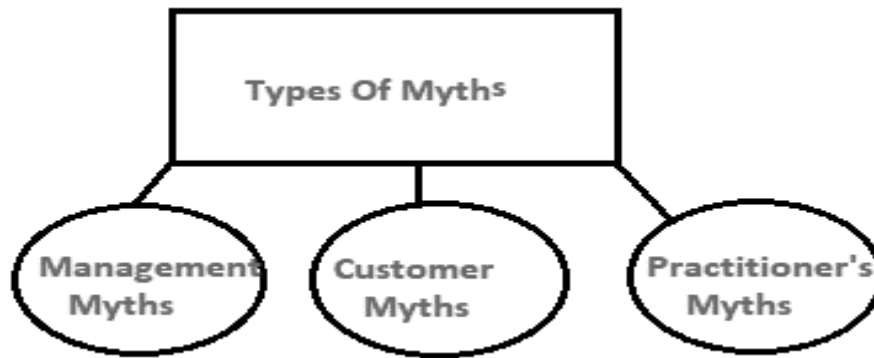
- *Is it possible to test each component part of the solution?* Has a reasonable testing strategy been implemented?
- *Does the solution produce results that conform to the data, functions, features, and behavior that are required?* Has the software been validated against all stakeholder requirements?



## SOFTWARE MYTHS

### Software Myths:

Most, experienced experts have seen myths or superstitions (false beliefs or interpretations) misleading attitudes (naked users) which creates. Major problems for management and technical people. The opposite Types of software-related myths are listed below.



### Types of Software Myths

#### (i) Management Myths:

##### Myth 1:

We have all the standards and procedures available for software development i.e. the software developer has all the reqd.

Fact:

- Software experts do not know that there are all of them levels.
- Such practices may or may not be expired at present / modern software engineering methods.
- And all existing processes are incomplete.

##### Myth 2:

The addition of the latest hardware programs will improve the software development.

Fact:

- The role of the latest hardware is not very high on standard software development; instead (CASE) Engineering tools help the computer they are more important than hardware to produce quality and productivity.
- Hence, the hardware resources are misused.

##### Myth 3:

- Managers think that, with the addition of more people and program planners to Software development can help meet project deadlines (If lagging behind).

Fact:

- Software development is not, the process of doing things like production; here the addition of people in previous stages can reduce the time it will be used for productive development, as the newcomers would take time existing developers of definitions and understanding of the file project. However, planned additions are organized and organized It can help complete the project.



## Different Stages of Myths

### (ii)Customer Myths:

The customer can be the direct users of the software, the technical team, marketing / sales department, or other company. Customer has myths

Leading to false expectations (customer) & that's why you create dissatisfaction with the developer.

#### Myth 1:

A general statement of intent is enough to start writing plans (software development) and details of objectives can be done over time.

Fact:

- Official and detailed description of the database function, ethical performance, communication, structural issues and the verification process are important.
- It is happening that the complete communication between the customer and the developer is required.

#### Myth 2:

- Project requirements continue to change, but, change, can be easy location due to the flexible nature of the software.

Fact:

- Changes were made to the final stages of software development but cost to make those changes grow through the latest stages of Development. A detailed analysis of user needs should be done to minimize change requirement. Figure shows the transition costs in Respect of the categories of development.

### (iii)Practitioner's Myths:

#### Myths 1:

They believe that their work has been completed with the writing of the plan and they received it to work.

Fact:

- It is true that every 60-80% effort goes into the maintenance phase (as of the latter software release). Efforts are required, where the product is available first delivered to customers.

#### Myths 2:

There is no other way to achieve system quality, behind it done running.

Fact:

- Systematic review of project technology is the quality of effective software verification method. These updates are quality filters and more accessible than test.

### **Myth 3:**

An operating system is the only product that can be successfully exported project.

Fact:

- A working system is not enough, it is just the right document brochures and booklets are also reqd. To provide for guidance & software support.

### **Myth4:**

Engineering software will enable us to build powerful and unnecessary document & always delay us.

Fact:

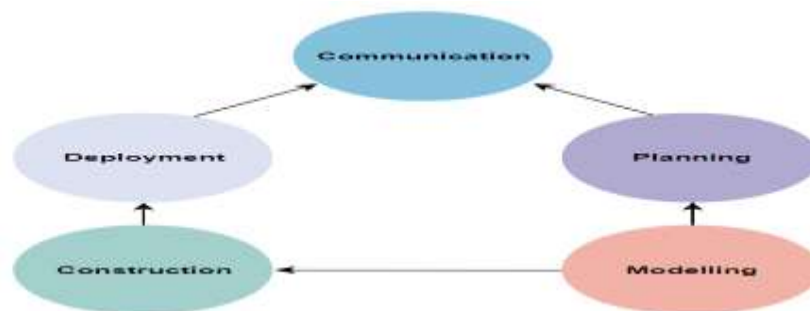
- Software engineering does not deal with text building, rather while creating better quality leads to reduced recycling & this is being studied for rapid product delivery.

## **A GENERIC PROCESS MODEL**

The **generic process model** is an abstraction of the software development process. It is used in most software since it provides a base for them.

The generic process model encompasses the following five steps:

1. Communication
2. Planning
3. Modelling
4. Construction
5. Deployment



### **Communication**

In this step, we communicate with the clients and end-users.

- We discuss the requirements of the project with the users.
- The users give suggestions on the project. If any changes are difficult to implement, we work on alternative ideas.

## **Planning**

In this step, we plan the steps for project development. After completing the final discussion, we report on the project.

- Planning plays a key role in the software development process.
- We discuss the risks involved in the project.

## **Modelling**

In this step, we create a model to understand the project in the real world. We showcase the model to all the developers. If changes are required, we implement them in this step.

- We develop a practical model to get a better understanding of the project.

## **Construction**

In this step, we follow a procedure to develop the final product.

- If any code is required for the project development, we implement it in this phase.
- We also test the project in this phase.

## **Deployment**

In this phase, we submit the project to the clients for their feedback and add any missing requirements.

- We get the client feedback.
- Depending on the feedback form, we make the appropriate changes.

## **PROCESS ASSESSMENT AND IMPROVEMENT**

Software processes are assessed to ensure their ability to control the cost, time and quality of software. Assessment is done to improve the software process followed by an organization.

Software Process Improvement (SPI) Cycle includes:

- Process measurement
- Process analysis
- Process change

## **Different approaches towards process assessment include**

CMM (Capability Maturity Model) and CMMI (Capability Maturity Model Integration)

CMM was developed by SEI (Software Engineering Institute) and evolved into CMMI later. It is an approach based on which an organization's process maturity is determined.

CMM's **five** maturity levels:

1. **Initial Level:** Processes are not organized and the success of a project depends only on the competence of the individual working on it. May not be able to repeat past successes in future projects. The probability of exceeding the estimated cost and schedule is high.
2. **Repeatable Level:** In this level, successes of the past could be repeated because the organization uses project management techniques to track cost and schedule. Management according to a documented plan helps in the improved process.
3. **Defined Level:** Organization's set of standard processes are defined and are slightly modified to incorporate each project demands. This provides consistency throughout the works of the organization.
4. **Managed Level:** Management of processes using quantitative techniques improves performance. Processes are assessed through data collection and analysis.
5. **Optimizing Level:** Processes are monitored and improved through feedback from current work. Innovative techniques are applied to cope with changing business objectives and the environment.

**CMMI** maturity levels include:

- Initial.
- Managed.
- Defined.
- Quantitatively Managed.
- Optimized.

**CMMI** capability levels include:

- **Level 0: Incomplete** – Incomplete processes are processes that are not performed or partially performed.
- **Level 1: Performed** – Specific goals are satisfied by processes and yet certain objectives related to quality, cost and schedule are not met. Useful work can be done.
- **Level 2: Managed** – Cost, quality and schedule are managed and processes are monitored by management techniques.
- **Level 3: Defined** – It includes management and additionally follow the organization's specified set of standard processes which are altered for each project.
- **Level 4: Quantitatively Managed** – Statistical and quantitative techniques are used for the management of processes.
- **Level 5: Optimized** – It focuses on continuous improvement of Quantitatively Managed process through innovations and nature of processes.

M.Srikanth,  
M.Tech., (Ph.D)  
Asst.Prof, Dept of IT, VIT

## **Standard CMMI Appraisal Method for Process Improvement (SCAMPI)**

It is a method used by Software Engineering Institute (SEI) for providing quality ratings with respect to Capability Maturity Model Integration (CMMI). Assessment includes five phases initiating, diagnosing, establishing, acting and learning. The appraisal process includes preparation, on-site activities, findings and ratings, final reporting etc.

## **CMM Based Appraisal for Internal Process Improvement (CBA IPI)**

It is an SEI CMM (Capability Maturity Model) based assessment method that provides diagnostics, enables and encourages an organization to understand its maturity. It gives the organization an insight into its software development capability by assessing the strength and weakness of the current process.

## **SPICE(ISO/IEC15504)**

This standard is one of the joint missions of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). They assist organizations in developing an objective evaluation of the effectiveness of a software process and related business management functions.

It consists of six levels as:

- Not performed.
- Performed informally.
- Planned and tracked.
- Well defined.
- Quantitatively controlled.
- Continuously improved.

## **ISO 9001:2000 for software**

It is applied to organizations aiming to improve the overall quality of product, process and services. They evaluate the ability of an organization to consistently provide products that meet customer requirements. Here the main aim of the organization should be enhancing customer satisfaction.

It follows **Plan Do Check Act** (PDCA) cycle which includes:

- **Planning** by defining the processes and their needs required to develop a better-quality product.
- **Doing** necessary actions according to the plan.
- **Checking** whether the actions for ensuring quality according to requirements are fulfilled.
- **Acting** on activities that are used to improve processes in the organization.

## PREScriptive PROCESS MODELS

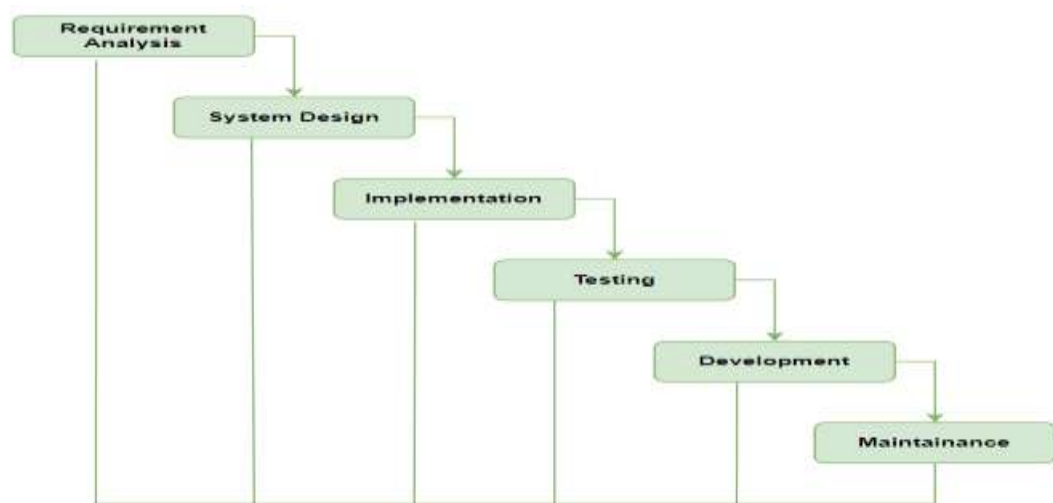
Prescriptive process models prescribe a set of framework and other activities, quality assurance points, and software process-related elements. They define a workflow among these elements that shows their inter-relationship.

The process models described here are,

- Waterfall Model.
- Incremental Process Model.
- RAD model

### 1. The Waterfall Model

- The waterfall model is also called as '**Linear sequential model**' or '**Classic life cycle model**'.
- In this model, each phase is fully completed before the beginning of the next phase.
- This model is used for the small projects.
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is complete.



**Figure: The Waterfall Model**



### **Advantages of waterfall model**

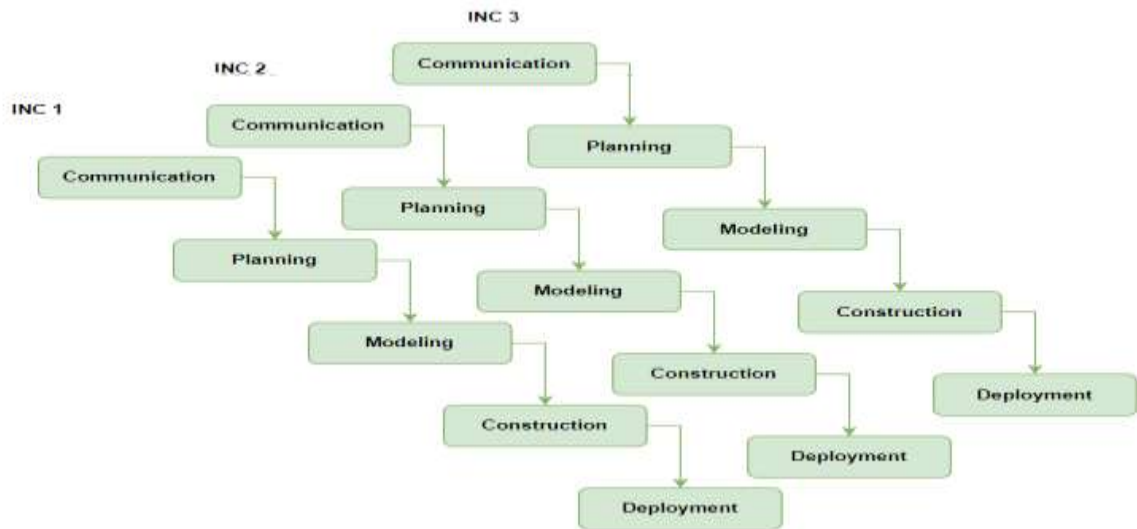
- The waterfall model is simple and easy to understand, implement, and use.
- All the requirements are known at the beginning of the project; hence it is easy to manage.
- It avoids overlapping of phases because each phase is completed at once.
- This model works for small projects because the requirements are understood very well.
- This model is preferred for those projects where the quality is more important as compared to the cost of the project.

### **Disadvantages of the waterfall model**

- This model is not good for complex and object-oriented projects.
- It is a poor model for long projects.
- The problems with this model are uncovered, until the software testing.
- The amount of risk is high.

## **2. Incremental Process model**

- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for any suggested modifications.
- The next increment implements on the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is finished.  
**For example,** the word-processing software is developed using the incremental model.



**Figure: Incremental Process model**

### **Advantages of incremental model**

- This model is flexible because the cost of development is low and initial product delivery is faster.
- It is easier to test and debug during the smaller iteration.
- The working software generates quickly and early during the software life cycle.
- The customers can respond to its functionalities after every increment.

### **Disadvantages of the incremental model**

- The cost of the final product may cross the cost estimated initially.
- This model requires a very clear and complete planning.
- The planning of design is required before the whole system is broken into small increments.
- The demands of customer for the additional functionalities after every increment causes problem during the system architecture.

### **3. RAD model**

- RAD is a Rapid Application Development model.
- Using the RAD model, software product is developed in a short period of time.
- The initial activity starts with the communication between customer and developer.
- Planning depends upon the initial requirements and then the requirements are divided into groups.
- Planning is more important to work together on different modules.

**The RAD model consist of following phases:**

### **1. Business Modeling**

- Business modeling consist of the flow of information between various functions in the project.
- For example what type of information is produced by every function and which are the functions to handle that information.
- A complete business analysis should be performed to get the essential business information.

### **2. Data modelling**

- The information in the business modelling phase is refined into the set of objects and it is essential for the business.
- The attributes of each object are identified and define the relationship between objects.

### **3. Process modelling**

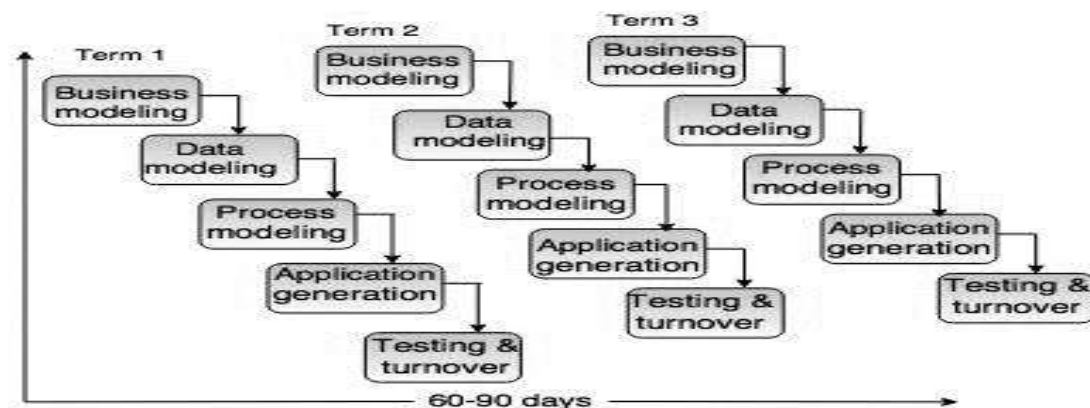
- The data objects defined in the data modelling phase are changed to fulfil the information flow to implement the business model.
- The process description is created for adding, modifying, deleting or retrieving a data object.

### **4. Application generation**

- In the application generation phase, the actual system is built.
- To construct the software the automated tools are used.

### **5. Testing and turnover**

- The prototypes are independently tested after each iteration so that the overall testing time is reduced.
  - The data flow and the interfaces between all the components are are fully tested.
- Hence, most of the programming components are already tested.



**Fig. - RAD Model**

## SPECIALIZED PROCESS MODELS

### Component-Based Development (CBD)

The component Based Development Model has the characteristics of a spiral model, hence is evolutionary and iterative in nature. In this model, applications are built from pre-packaged software components that are available from different vendors. Components are modular products with well-defined functions that can be incorporated into the project of selection. The modelling and construction stages are begun with identifying candidate components suitable for the project.

Steps involved in this approach are implemented in an evolutionary fashion:

1. Components suitable for the application domain are selected.
2. Component integration issues are catered to.
3. Software architecture is designed based on the selected components.
4. Components are integrated into the architecture.
5. Testing of the functionality of components is done.

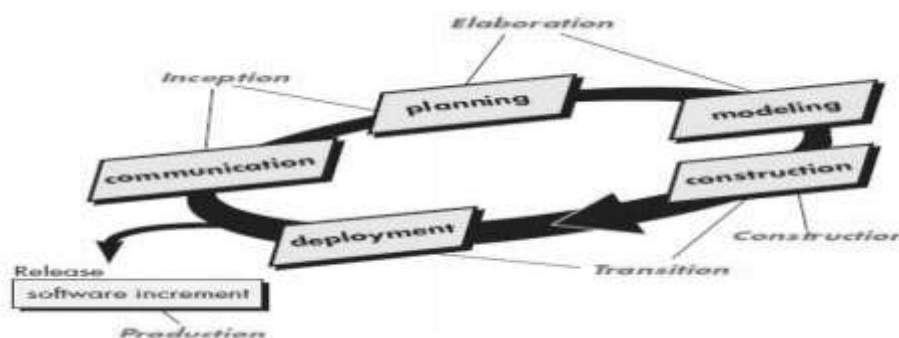
Software can be re-used in this manner, cost and time is reduced.

### Formal Methods Model (FMM)

In the Formal Methods Model, mathematical methods are applied in the process of developing software. It Uses Formal Specification Language (FSL) to define each system characteristics. FSL defines the syntax, notations for representing system specifications, several objects and relations to define the system in detail. The careful mathematical analysis that is done in FMM results in a defect-free system. It also helps to identify and correct ambiguity and inconsistency easily. This method is time-consuming and expensive in nature. Also, knowledge of formal methods is necessary for developers and is a challenge.

## THE UNIFIED PROCESS

Unified process (UP) is an architecture centric, use case driven, iterative and incremental development process. UP is also referred to as the unified software development process.



### The Unified Process in Software Engineering

M.Srikanth,  
M.Tech., (Ph.D)  
Asst.Prof, Dept of IT, VIT

The Unified Process is an attempt to draw on the best features and characteristics of traditional software process models, but characterize them in a way that implements many of the best principles of agile software development. The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system. It emphasizes the important role of software architecture and “helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse”. It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

### **Phases of the Unified Process**

This process divides the development process into five phases:

- Inception
- Elaboration
- Conception
- Transition
- Production

**Inception:** establish the business case for the system, define risks, obtain 10% of the requirements, estimate next phase effort.

**Elaboration:** Develop an understanding of the problem domain and the system architecture, risk significant portions may be coded/tested, 80% major requirements identified.

**Construction:** system design, programming and testing. Building the remaining system in short iterations.

**Transition:** Deploy the system in its operating environment. Deliver releases for feedback and deployment.

## **PERSONAL AND TEAM PROCESS MODELS**

The best software process is personal and team process model one that is close to the people who will be doing the work. Watts Humphrey proposed two process models. Models “Personal Software Process (PSP)” and “Team Software Process (TSP).” Both require hard work, training, and coordination, but both are achievable.

### **Personal Software Process (PSP)**

The Personal Software Process (PSP) emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product.

In addition, PSP makes the practitioner responsible for project planning and empowers the practitioner to control the quality of all software work products that are developed. The PSP model defines five framework activities:

- **Planning.** This activity isolates requirements and develops both size and resource estimates. In addition, defects estimate (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.
- **High level design.** External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.
- **High level design review.** Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.
- **Development.** The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.
- **Postmortem.** Using the measures and metrics collected, the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

PSP stresses the need to identify errors early and, just as important, to understand the types of errors that you are likely to make. PSP represents a disciplined, metrics based approach to software engineering that may lead to culture shock for many practitioners.

### **Team Software Process (TSP)**

Watts Humphrey extended the lessons learned from the introduction of PSP and proposed a Team Software Process (TSP). The goal of TSP is to build a “self-directed” project team that organizes itself to produce high quality software.

### **Humphrey defines the following objectives for TSP:**

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPTs) of 3 to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance

- Accelerate software process improvement by making CMM23 Level 5 behavior normal and expected.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.

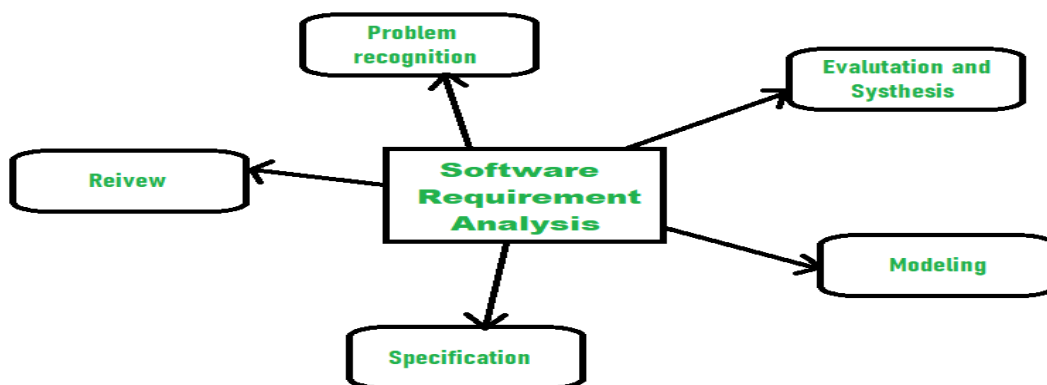
A self-directed team has a consistent understanding of its overall goals and objectives; defines roles and responsibilities for each team member; tracks quantitative project data (about productivity and quality); identifies a team process that is appropriate for the project and a strategy for implementing the process; defines local standards that are applicable to the team's software engineering work; continually assesses risk and reacts to it; and tracks, manages, and reports project status. TSP defines the following framework activities: project launch, high level design, implementation, personal and team process model, integration and test, and post-mortem. TSP makes use of a wide variety of scripts, forms, and standards that serve to guide team members in their work. "Scripts" define specific process activities (project launch, design, implementation, integration and system testing, post-mortem) and other more detailed work functions (development planning, requirements development, software configuration management, unit test) that are part of the team process.

## UNIT-II

### Requirements Analysis and Specification

#### REQUIREMENTS ANALYSIS

Software requirement analysis simply means complete study, analyzing, describing software requirements so that requirements that are genuine and needed can be fulfilled to solve problem. There are several activities involved in analyzing Software requirements. Some of them are given below:



1. **Problem Recognition** : The main aim of requirement analysis is to fully understand main objective of requirement that includes why it is needed, does it add value to product, will it be beneficial, does it increase quality of the project, does it will have any other effect. All these points are fully recognized in problem recognition so that requirements that are essential can be fulfilled to solve business problems.
2. **Evaluation and Synthesis** : Evaluation means judgement about something whether it is worth or not and synthesis means to create or form something. Here are some tasks are given that is important in the evaluation and synthesis of software requirement :
  - To define all functions of software that necessary.
  - To define all data objects that are present externally and are easily observable.
  - To evaluate that flow of data is worth or not.
  - To fully understand overall behavior of system that means overall working of system.
  - To identify and discover constraints that are designed.
  - To define and establish character of system interface to fully understand how system interacts with two or more components or with one another.
3. **Modeling** : After complete gathering of information from above tasks, functional and behavioral models are established after checking function and behavior of system using a domain model that also known as the conceptual model.



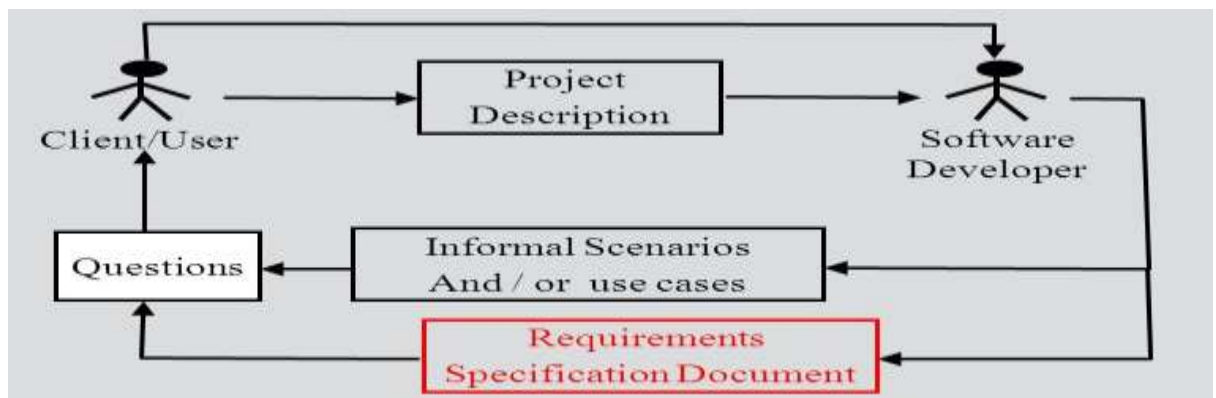
4. **Specification :** The software requirement specification (SRS) which means to specify the requirement whether it is functional or non-functional should be developed.
5. **Review :** After developing the SRS, it must be reviewed to check whether it can be improved or not and must be refined to make it better and increase the quality.

### REQUIREMENTS SPECIFICATION

A document that clearly and precisely describes each of the essential requirements of the software and the external interfaces (functions, performance, design constraint and quality attributes) Each requirement is defined in such a way that its achievement is capable of being objectively verified by a prescribed method for example inspection, demonstration, analysis or test.

### REQUIREMENTS GATHERING

- This activity involves interviews the end user and customer and studying the existing document to collect all possible information regarding the system.
- This is the process of finding out what a customer requires from a software system.
- It has the purpose to find out about the customer needs and to find out about inappropriate requirements, ambiguities etc.



### Requirements Gathering Process

- Identify the relevant stakeholders.
- Establish project goals and objectives.
- Produce requirements from stakeholders.
- Document the requirements.
- Confirm the requirements.
- Arrange the requirements.

## Requirements Analysis

- **Software requirement analysis** simply means complete study, analyzing, describing software requirements so that requirements that are genuine and needed can be fulfilled to solve problem. There are several activities involved in analyzing Software requirements.



- **Problem Recognition:** The main aim of requirement analysis is to fully understand the main objectives of requirements, including why they are needed, whether they will be beneficial, and whether they will increase the quality of the project. All of these things are fully thought through in the process of figuring out the problem, so that the needs that are most important can be met.
- **Evaluation and Synthesis:** Evaluation means judgement about something whether it is worth or not and synthesis means to create or form something. Here are some tasks are given that is important in the evaluation and synthesis of software requirement.
- **Modeling:** After complete gathering of information from above tasks, functional and behavioral models are established after checking function and behavior of system using a domain model that also known as the conceptual model.
- **Specification:** The software requirement specification (SRS) which means to specify the requirement whether it is functional or non-functional should be developed.
- **Review:** After developing the SRS, it must be reviewed to check whether it can be improved or not and must be refined to make it better and increase the quality.

## **SOFTWARE REQUIREMENT SPECIFICATION (SRS)**

- SRS stands for software requirement specification it is a document prepared by business analyst or system analyst.
- It describes what will be the features of software and what will be its behavior i.e., how it will perform.
- It is the details description of software system to be developed with its functional and non-functional requirement.
- The SRS consists of all necessary requirements required for the project development. In order to get all the details of software from customer and to write the SRS document system analyst is required.
- SRS document is actually an agreement between the customer and developer.

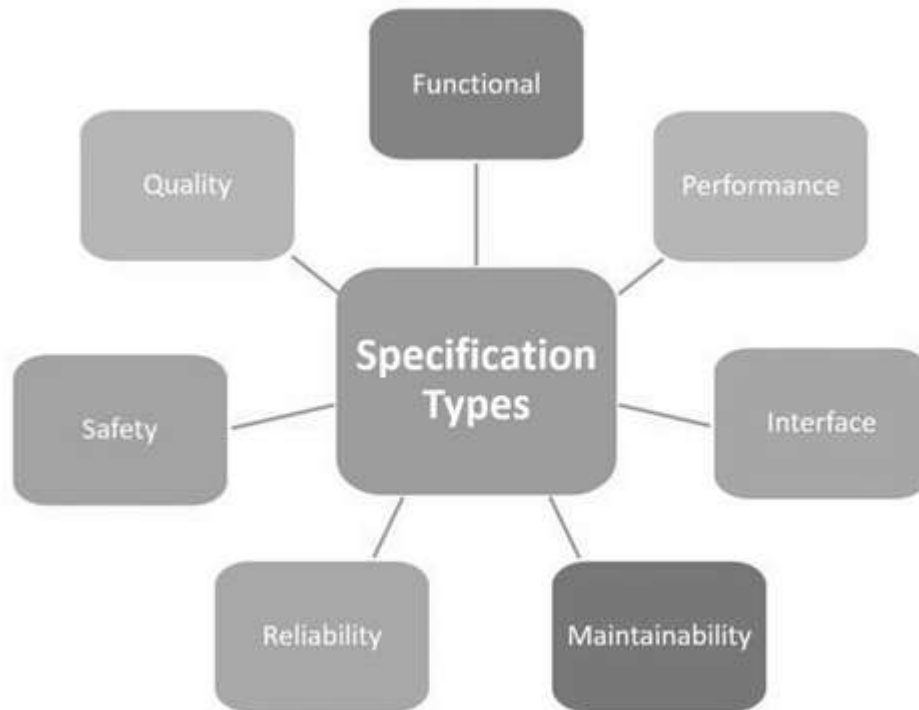
**Reason for using SRS:** describe how software system should be developed. It provides everyone involved with a roadmap for that project.

### **Characteristic of good SRS document.**

- **Concise:** The SRS document is to the point at the same time unambiguous, consistent and complete.
- **Structured:** The SRS document should be well structured.
- **Black box view:** The SRS should specify the externally visible behavior of the system.
- **Conceptual integrity:** The SRS document should exhibit conceptual integrity so that the reader can easily understand the contents
- **Verifiable:** whether or not requirement have been met in an implementation.

### **Components of SRS Document:**

- Functional Requirement
- Nonfunctional requirements
- Static requirements
- Standards to be followed
- Security requirements
- Company policies



## FORMAL SYSTEM SPECIFICATION

A formal technique is a mathematical method to specify a hardware and/or software system, verify whether a specification is realisable, verify that an implementation satisfies its specification, prove properties of a system without necessarily running the system, etc. The mathematical basis of a formal method is provided by its specification language. More precisely, a formal specification language consists of two sets —syn and sem, and a relation sat between them. The set syn is called the syntactic domain, the set sem is called the semantic domain, and the relation sat is called the satisfaction relation.

**Syntactic domains:** The syntactic domain of a formal specification language consists of an alphabet of symbols and a set of formation rules to construct well-formed formulas from the alphabet. The well-formed formulas are used to specify a system.

**Semantic domains:** Formal techniques can have considerably different semantic domains. Abstract data type specification languages are used to specify algebras, theories, and programs. Programming languages are used to specify functions from input to output values.

## SOFTWARE DESIGN

Software design is a procedure to change client prerequisites into some reasonable structure, which helps the developer in programming coding and execution. For evaluating client prerequisites, an SRS (Software Requirement Specification) archive is made though, for coding and usage, there is a need for progressively explicit and point by point necessities in programming terms. The yield of this procedure can straightforwardly be utilized into usage in programming dialects. Software design is the initial phase in SDLC (Software Design Life Cycle), which moves the focus from issue area to arrangement space. It attempts to indicate how to satisfy the necessities referenced in SRS. Process is a set of related and sequenced task that transform a set of input and set of output

### Objectives

- **Correctness:** software design should be correct as per requirement
- **Completeness:** design should have all components like: Data structure, Module, External interface etc.
- **Flexibility:** able to modify on changing needs
- **Consistency:** There should not any inconsistency in the design
- **Maintainability:** Easily maintainable by other design

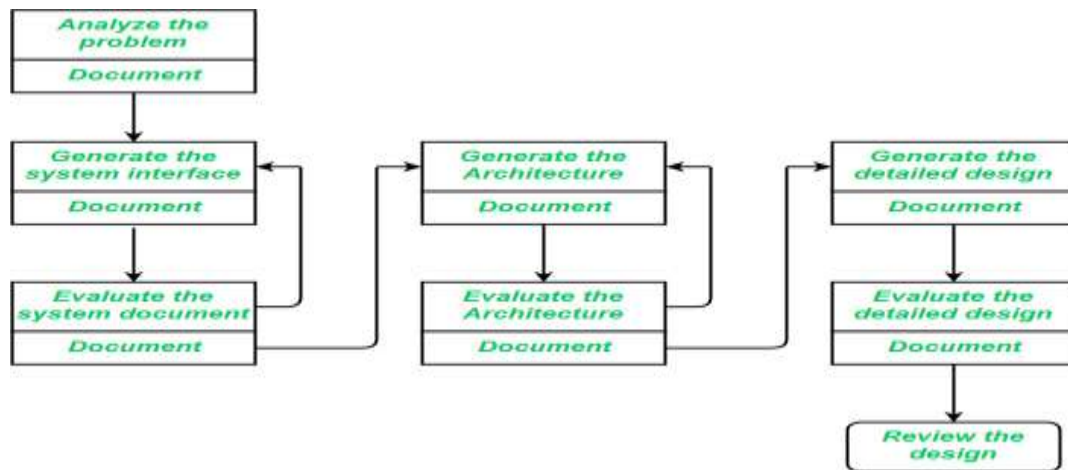
### Levels of software Design

- **Internal:** Focus on deciding which modules are needed
- **External:** Focus on planning and specify the internal structure and processing details

## OVERVIEW OF THE DESIGN PROCESS

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language. The software design process can be divided into the following three levels of phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design



**Interface Design:** *Interface design* is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored and the system is treated as a black box. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called *agents*.

Interface design should include the following details:

- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification on the data, and the formats of the data coming into and going out of the system.
- Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

**Architectural Design:** *Architectural design* is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored.

Issues in architectural design includes:

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces

- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

**Detailed Design:** *Design* is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.

The detailed design may include:

- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- User interfaces
- Unit states and state changes
- Data and control interaction between units
- Data packaging and implementation, including issues of scope and visibility of program elements
- Algorithms and data structures

## COHESION AND COUPLING

Cohesion and coupling are very important in software engineering. Cohesion and coupling are used to measure the modules.

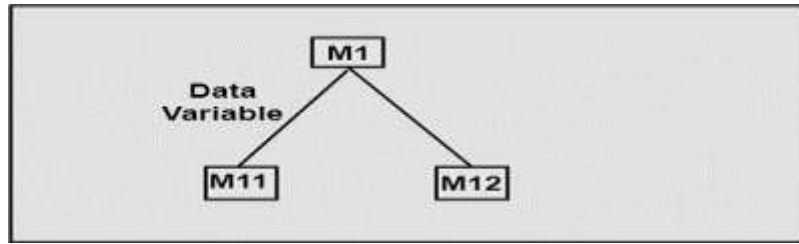
**Coupling:** coupling is used to measure the interdependent between the module.

Types of coupling:

They are 5 types of coupling are present

- Data coupling
- Stamp coupling
- Control coupling
- Common coupling
- Content coupling

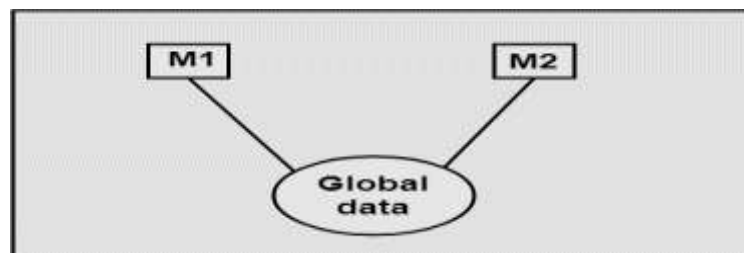
**1. Data Coupling:** The data is transformed to one module to another module. The dependence between the two modules is very low.



**2. Stamp Coupling:** The complete data can be transformed one module to another module is called stamp coupling.

**3. Control Coupling:** Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

**4. Common Coupling:** Two modules are common coupled if they share information through some global data items.



**5. Content Coupling:** Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

## COHESION

In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

Cohesion is an **ordinal** type of measurement and is generally described as "high cohesion" or "low cohesion."

### Types of Modules Cohesion

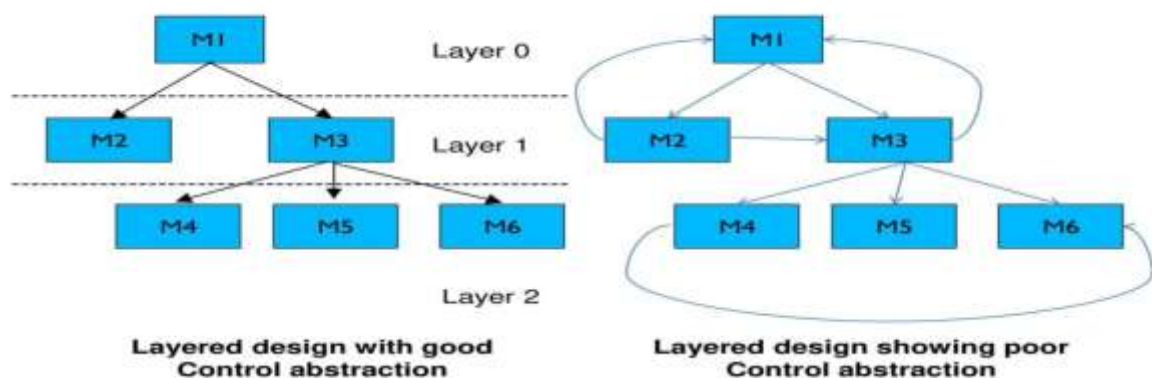
- 1. Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.



2. **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module forms the components of the sequence, where the output from one component of the sequence is input to the next.
3. **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.
4. **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.
5. **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.

### LAYERED ARRANGEMENT OF MODULES

- Graphical representation of call relations and form hierarchy of layers. Lower layer is unaware of the higher layer module.
- In the layered design, the modules are arranged in the hierarchy of layers. In such design, a module can only invoke functions of the modules in the layer immediately below it. The higher layer module can be considered similar to management who can invoke lower layer module to get a certain task done.



## APPROACHES TO SOFTWARE DESIGN

There are two generic approaches for software designing:


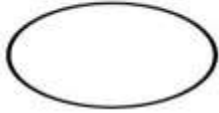


**Top-down Design** We know that a system is composed of more than one sub-system and it contains a number of components. Further, these sub-systems and components may have their own set of sub-system and components and creates hierarchical structure in the system. Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved. Top-down design starts with a generalized model of system and keeps on defining the more specific part of it. When all components are composed the whole system comes into existence. Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

### **Bottom-up Design**

The bottom-up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower-level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased. Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system. Both, top-down and bottom-up approaches are not practical individually. Instead, a good combination of both is used.

## **Developing the DFD Model of a System**

Data Flow Diagram (DFD) of a system represents how input data is converted to output data graphically. Level 0 also called context level represents most fundamental and abstract view of the system. Subsequently other lower levels can be decomposed from it. DFD model of a system contains multiple DFDs but there is a single data dictionary for entire DFD model. Data dictionary comprises definitions of data items used in DFD.

Symbol	Name	Function
	Data flow	Used to Connect Processes to each other, to sources or Sinks; the arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System Inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

**Symbols for Data Flow Diagrams**

**Context diagram:** It demonstrates entire data flow of a system in a single process/ bubble. Bubble is annotated with 'Noun' representing whole system. This is only bubble in DFD where a noun, (in the form of name of a system) is used. It is named since purpose of context diagram is to grab the context of system and not functionality. All other bubbles have a verb according to main function performed by it.

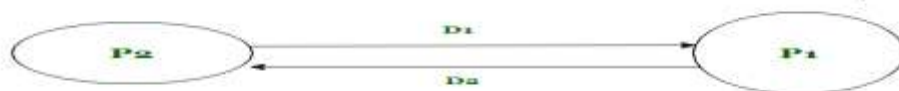
Context diagram shows three main things: users, data flow to system and from system. It captures various external entities interacting with system, data to and from system as incoming and outgoing arrows. Context diagram requires analysis of SRS document. Data flow is represented with data names on top of arrow.

**Construction of Level 1 and other lower-level diagrams:** Level 1 DFD contains 3 to 7 bubbles representing functions. There are processes that can be broken down into sub-process each representing a bubble. Every Bubble has a **verb** demonstrating functionality of that process. To create level 1/level 2 or other level DFDs, a bubble is broken into sub-parts containing 3-7 functionalities. If there are more than 7 sub-process then some related information can be combined together. If there are less than 3 sub-processes, it can be further decomposed to create more bubbles.

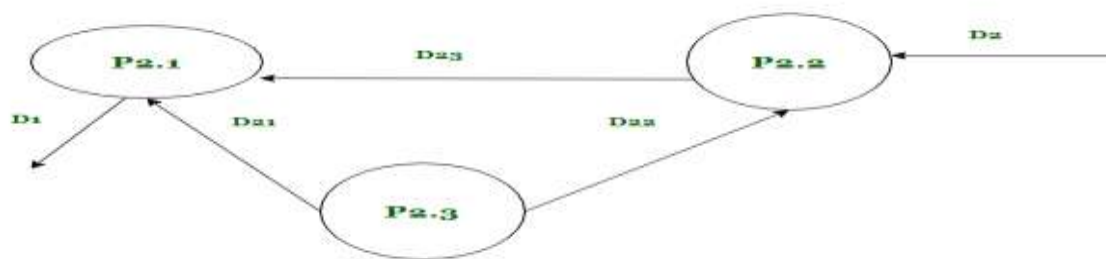
**Decomposition:** Bubbles are decomposed into sub functions at successive levels of DFD level. Decomposition is called exploding/factoring a bubble. Each bubble at any level can be broken to anything between 3 and 7 bubbles. But a bubble should not be decomposed further once it is found to represent simple set of instructions. Too many bubbles at any level makes dfd hard to understand. Very less bubble makes decomposition redundant and trivial.

**Numbering:** Each process symbol must utilize a unique reference number to differentiate from one other. When a bubble x is decomposed, its children are numbered as x.1, x.2 and so on. It can help determine its ancestors, successors, and precisely its level. For example, level 0 DFD is numbered as 0. Level 1 are numbered as 0.1, 0.2, 0.3 or 1, 2, 3 and so on. Level 2 are marked as 1.1, 1.2, 1.3 etc.

**Balancing DFD:** Parent dfd having inflow and out flow of data should match data flow at next child level. This is known as balancing a DFD. Concept is illustrated in figure:



*Level 1 DFD*



*Level 2 DFD decomposing P2*

In Level 1 DFD, data items D1 flow out of bubble 2 and item D2 flows into bubble 2. In next level, bubble 2 is decomposed into three sub process (2.1, 2.2, 2.3). It has data item D1 flowing out and D2 flowing in. So DFD is balanced here.

A bubble representing a process should have minimum one input and one output flow. When a bubble has input flow without any output flow, it is known as “black hole”. When a process has output flows but no input flows, it is called a “miracle”. A processing step may have outputs that are greater than sum of its inputs is called Grey holes.

#### **Common mistakes to avoid while constructing DFDs :**

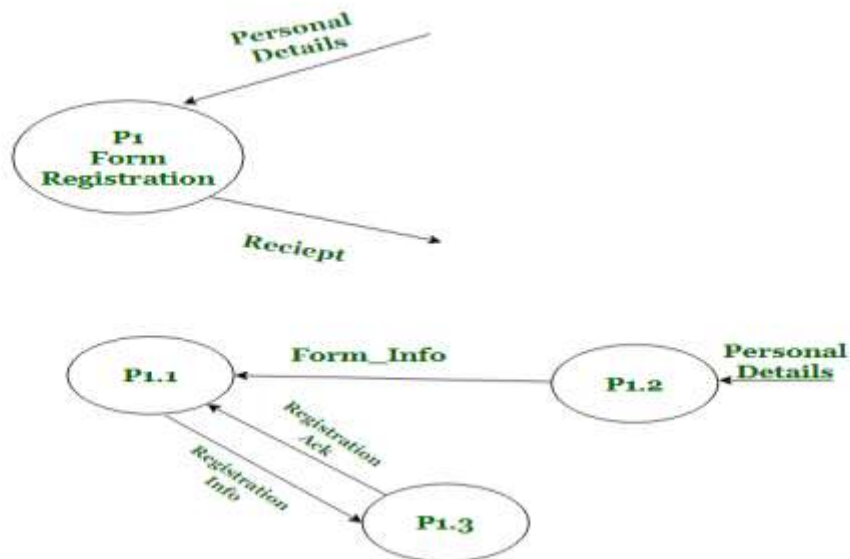
- DFDs should always represent data flow and there should be no control flow.
- All external entities should be represented at context level.
- All functionality of system must be captured in dfd and none should be overlooked. Also, only those functions specified in SRS should be represented.
- Arrows connecting to data store need not be annotated with any data name.

M.Srikanth,  
M.Tech., (Ph.D)  
Asst.Prof, Dept of IT, VIT



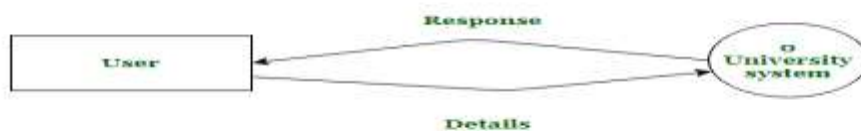
*DataStore arrow has no annotation*

- DFDs should always be balanced at all levels. Dataflow in and out of parent process must be present in child diagram.



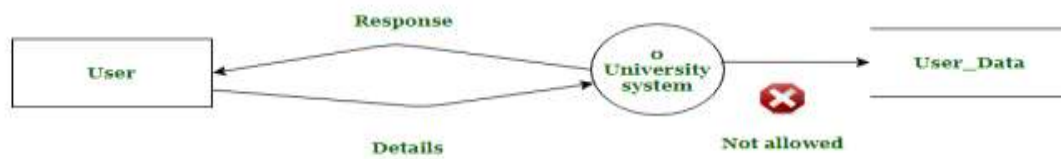
*Unbalanced DFD (No receipt generation in child)*

- Context level bubble always contains name of entire system in form of noun and no verb can be used in its representation. Whereas all other levels only have verb in bubble representation.



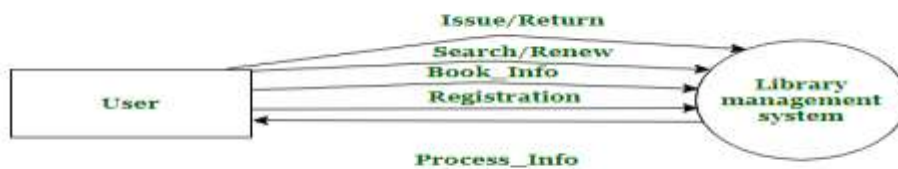
*Context Level having Noun as system name*

- There should not be any database in level 0. Level 0 contains only one bubble and external entities.



### *Incorrect Level 0 with datastore*

- No cluttering should be depicted in DFD. When too many data flowing occurs in and out of DFD, combine these data item into high level item?



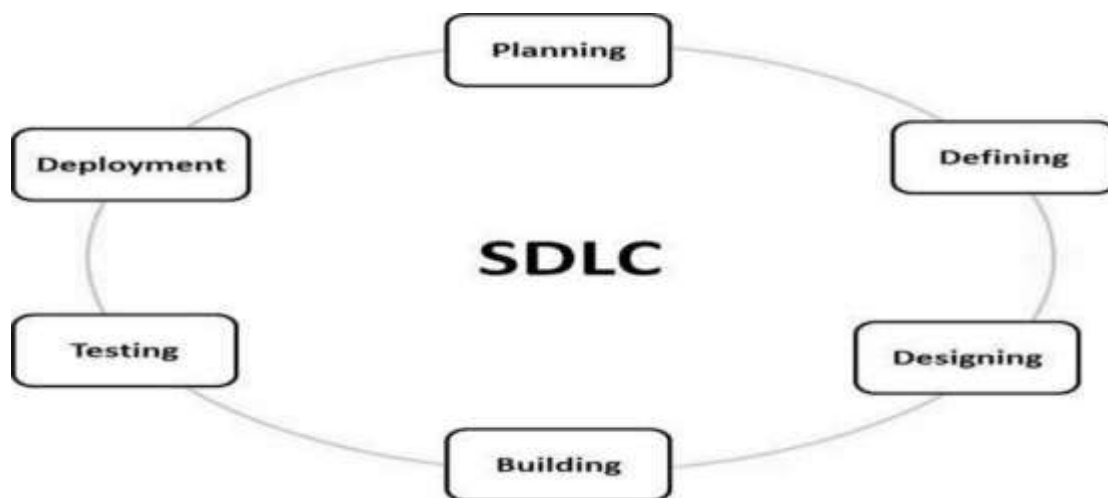
### *DFD with cluttering*

- All bubbles having unique process in DFDs should be connected to another process or to a data store. It cannot exist by itself, unconnected to rest of system.

## WHAT IS SDLC?

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development Life Cycle consists of the following stages –

### Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC's development. It is performed by the senior members of the team with inputs from the customer, market surveys and domain experts. This information is then used to plan the basic project approach and conduct product feasibility study.

### Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

### Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Usually more than one design approach is proposed and

documented in a DDS - Design Document Specification. The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

#### **Stage 4: Building or Developing the Product**

The programming code is generated as per DDS during this stage of development. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding.

#### **Stage 5: Testing the Product**

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

#### **Stage 6: Deployment in the Market and Maintenance**

The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing). Based on the feedback, the product may be released as it is or with suggested enhancements. After the product is released in the market, its maintenance is done for the existing customer base.

### **SDLC Models**

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as "Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry –

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model

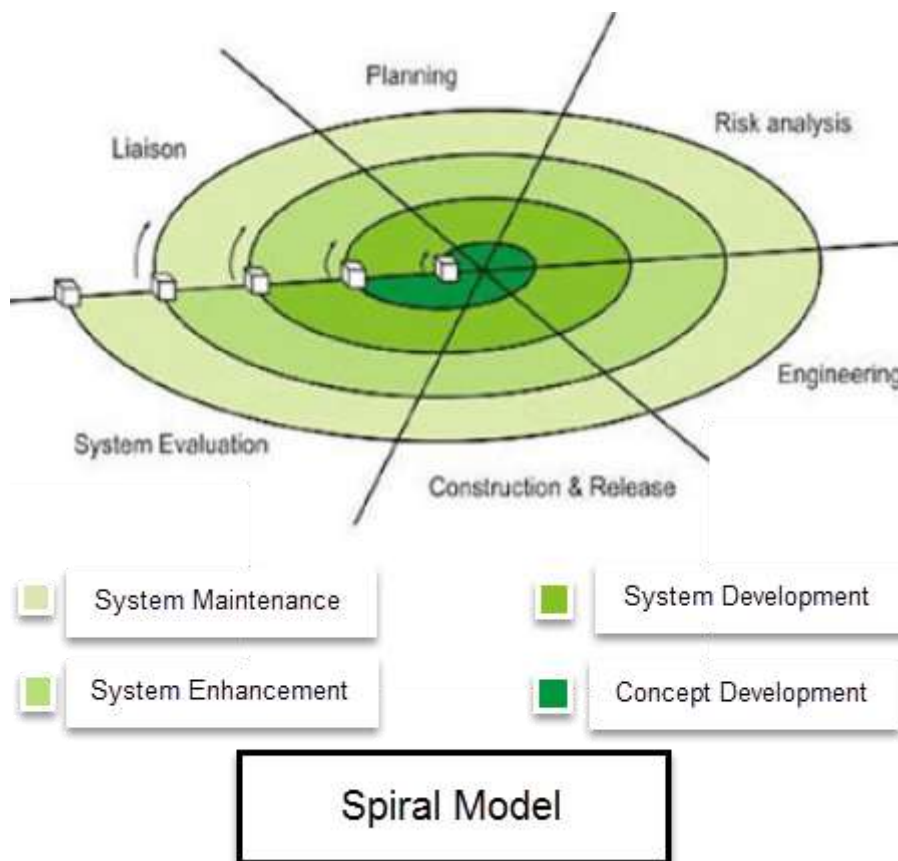
Other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.



## SPIRAL MODEL

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

The Spiral Model is shown in fig:



## Spiral Model Phases

<b>Spiral Model Phases</b>	<b>Activities performed during phase</b>
<b>Planning</b>	It includes estimating the cost, schedule and resources for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer
<b>Risk Analysis</b>	Identification of potential risk is done while risk mitigation strategy is planned and finalized
<b>Engineering</b>	It includes testing, coding and deploying software at the customer site
<b>Evaluation</b>	Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost overrun

### The advantages

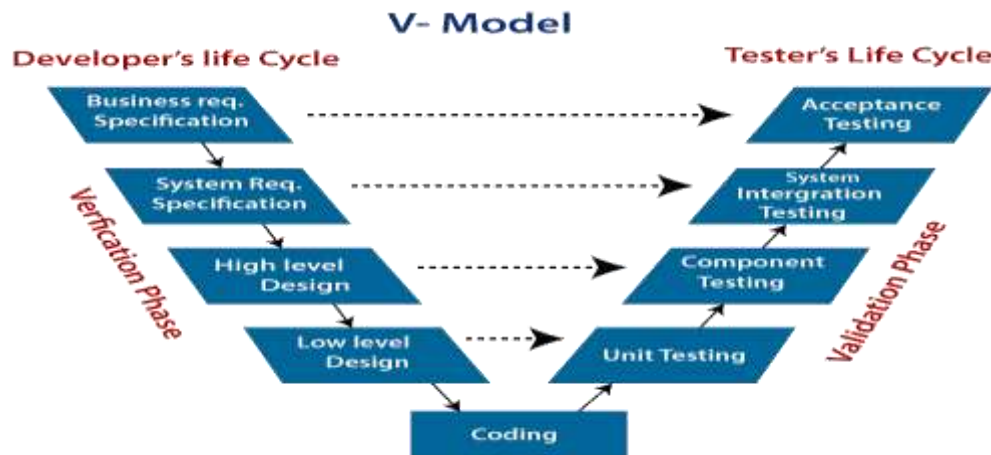
- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

### The disadvantages

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

## V-MODEL

V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.



**Verification:** It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

**Validation:** It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.

**There are the various phases of Verification Phase of V-model:**

1. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.

4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

**There are the various phases of Validation Phase of V-model:**

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client's business team. System Test ensures that expectations from an application developer are met.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

**V- Model — Application**

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

## **UNIT – III**

### **UNIFIED MODELING LANGUAGE (UML)**

#### **INTRODUCTION TO UML**

Way back in the late twentieth century -- 1997 to be exact -- the Object Management Group (OMG) released the Unified Modeling Language (UML). One of the purposes of UML was to provide the development community with a stable and common design language that could be used to develop and build computer applications. UML brought forth a unified standard modeling notation that IT professionals had been wanting for years. Using UML, IT professionals could now read and disseminate system structure and design plans -- just as construction workers have been doing for years with blueprints of buildings.

It is now the twenty-first century and UML has gained traction in our profession. On 75 percent of the resumes I see, there is a bullet point claiming knowledge of UML. However, after speaking with a majority of these job candidates, it becomes clear that they do not truly know UML. Typically, they are either using it as a buzz word, or they have had a sliver of exposure to UML. This lack of understanding inspired me to write this quick introduction to UML, focused on the basic diagrams used in visual modeling. When you are finished reading you will not have enough knowledge to put UML on your resume, but you will have a starting point for digging more deeply into the language.

#### **WHAT IS UML**

UML (Unified Modeling Language) is a general-purpose, graphical modeling language in the field of Software Engineering. UML is used to specify, visualize, construct, and document the artifacts (major elements) of the software system. It was initially developed by Grady Booch, Ivar Jacobson, and James Rumbaugh in 1994-95 at Rational software, and its further development was carried out through 1996. In 1997, it got adopted as a standard by the Object Management Group.

UML (Unified Modeling Language) is a general-purpose, graphical modeling language in the field of Software Engineering. UML is used to specify, visualize, construct, and document the artifacts (major elements) of the software system. It was initially developed by Grady Booch, Ivar Jacobson, and James Rumbaugh in 1994-95 at Rational software, and its further development was carried out through 1996. In 1997, it got adopted as a standard by the Object Management Group.

#### **WHY WE MODEL**

We build models to communicate the desired structure and behavior of our system. We build models to visualize and control the system's architecture. We build models to better understand the system we are building, often exposing opportunities for simplification and reuse. And we build models to manage risk

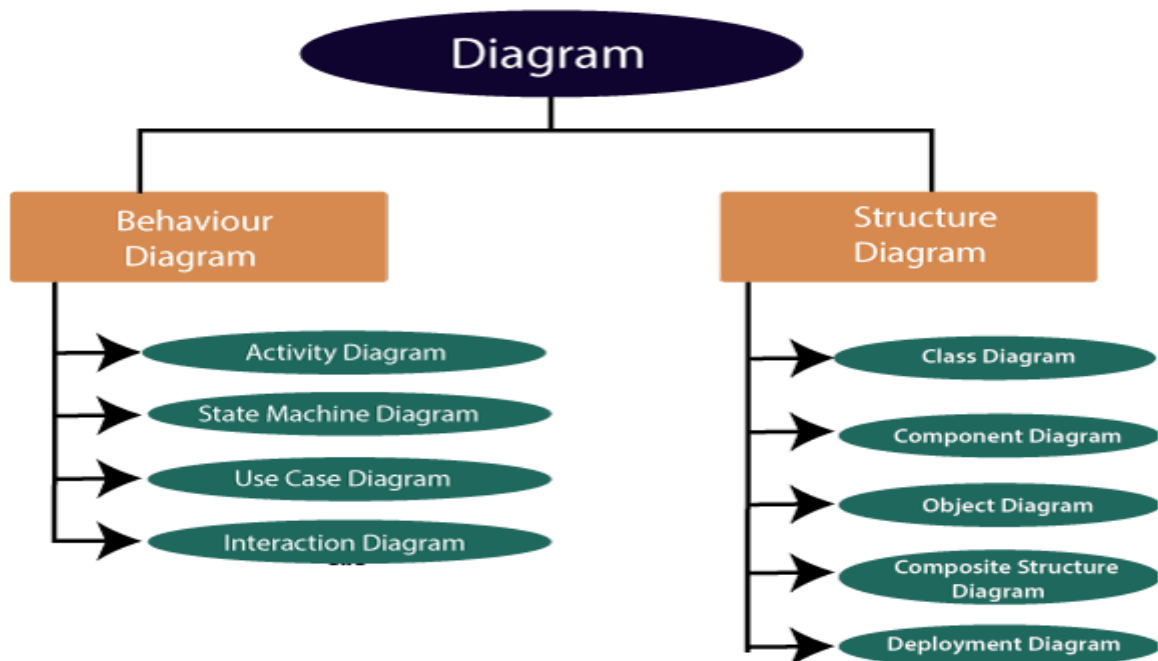
M.Srikanth,  
M.Tech., (Ph.D)  
Asst.Prof, Dept of IT, VIT

## STANDARD DIAGRAMS

- The elements are like components which can be associated in different ways to make a complete UML picture, which is known as diagram. Thus, it is very important to understand the different diagrams to implement the knowledge in real-life systems.
- Any complex system is best understood by making some kind of diagrams or pictures. These diagrams have a better impact on our understanding. If we look around, we will realize that the diagrams are not a new concept but it is used widely in different forms in different industries.
- We prepare UML diagrams to understand the system in a better and simple way. A single diagram is not enough to cover all the aspects of the system. UML defines various kinds of diagrams to cover most of the aspects of a system.
- You can also create your own set of diagrams to meet your requirements. Diagrams are generally made in an incremental and iterative way.

There are two broad categories of diagrams and they are again divided into subcategories –

- Structural Diagrams
- Behavioral Diagrams



## STRUCTURAL DIAGRAMS

Structural diagrams depict a static view or structure of a system. It is widely used in the documentation of software architecture. It embraces class diagrams, composite structure diagrams, component diagrams, deployment diagrams, object diagrams, and package diagrams. It presents an outline for the system.

Diagram Type	Detail
Class	Class diagrams capture the logical structure of the system, the Classes and objects that make up the model, describing what exists and what attributes and behavior it has.
Composite Structure	Composite Structure diagrams reflect the internal collaboration of Classes, Interfaces and Components (and their properties) to describe a functionality.
Component	Component diagrams illustrate the pieces of software, embedded controllers and such that make up a system, and their organization and dependencies.
Deployment	Deployment diagrams show how and where the system is to be deployed; that is, its execution architecture.
Object	Object diagrams depict object instances of Classes and their relationships at a point in time.

## CLASS DIAGRAM

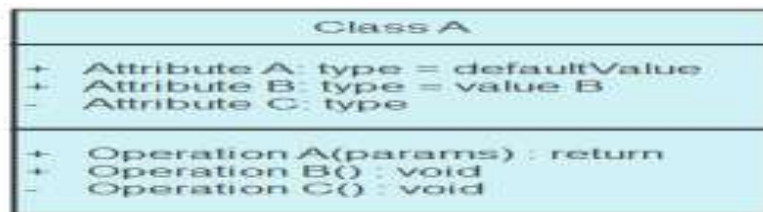
- A class diagram is a static structure that is used in software engineering. A class diagram shows the classes, attributes, operations, and the relationship between them. This helps software engineers in developing the code for an application. It is also used for describing, visualizing, and documenting different facets of a system.
- Class diagrams are the only UML diagrams that can be mapped directly with object-oriented languages. That is why they are frequently used in the modeling of object-oriented systems and are widely used during the construction of object-oriented systems.
- Class diagrams are one of the most important diagrams in coding as they form the basis for component and deployment diagrams and describe the responsibilities in a system. Along with

that, they are used for the analysis and design of an application and are also used in forward and reverse engineering.

## Class Notation

There are three major parts of a class diagram as shown in the image below:

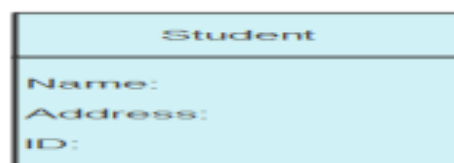
1. Class Name
2. Class Attributes
3. Class Operations



A single rectangle is used to represent the class as shown above. The rectangle is divided into three compartments with the topmost being Class Name, then Attributes in the middle, and Operations in the bottom.

**Class Name:** The class name is important for graphical representation. It should be written in bold in the top compartment and start with a capital letter. Moreover, an abstract class should be written in italics.

**Attributes:** Attributes are written in the middle compartment and list down all the properties of the object being modeled. You can simply add new attributes or derive new attributes from already listed attributes. Attributes must be meaningful and are usually used with the visibility factor that describes the accessibility of an attribute.

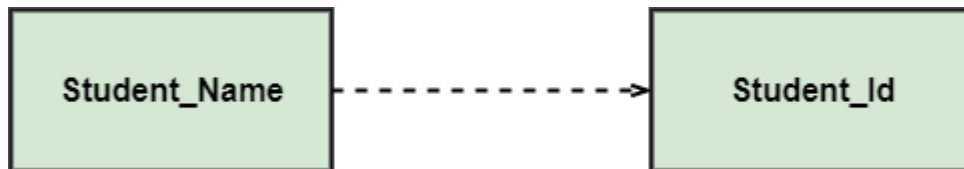


## Relationships

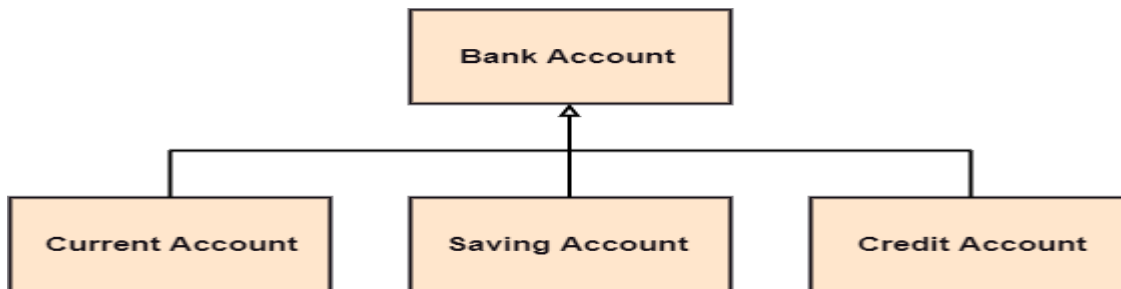
**In UML, relationships are of three types:**

**Dependency:** A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship. In the following example, Student\_Name is dependent on the Student\_Id.





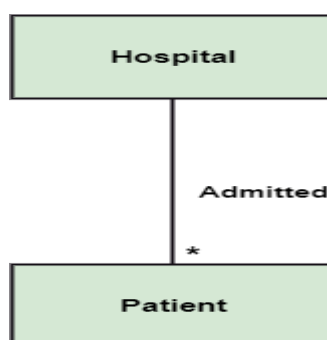
**Generalization:** A generalization is a relationship between a parent class (superclass) and a child class (subclass). In this, the child class is inherited from the parent class. For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.



**Association:** It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship. For example, a department is associated with the college.



**Multiplicity:** It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity. For example, multiple patients are admitted to one hospital.



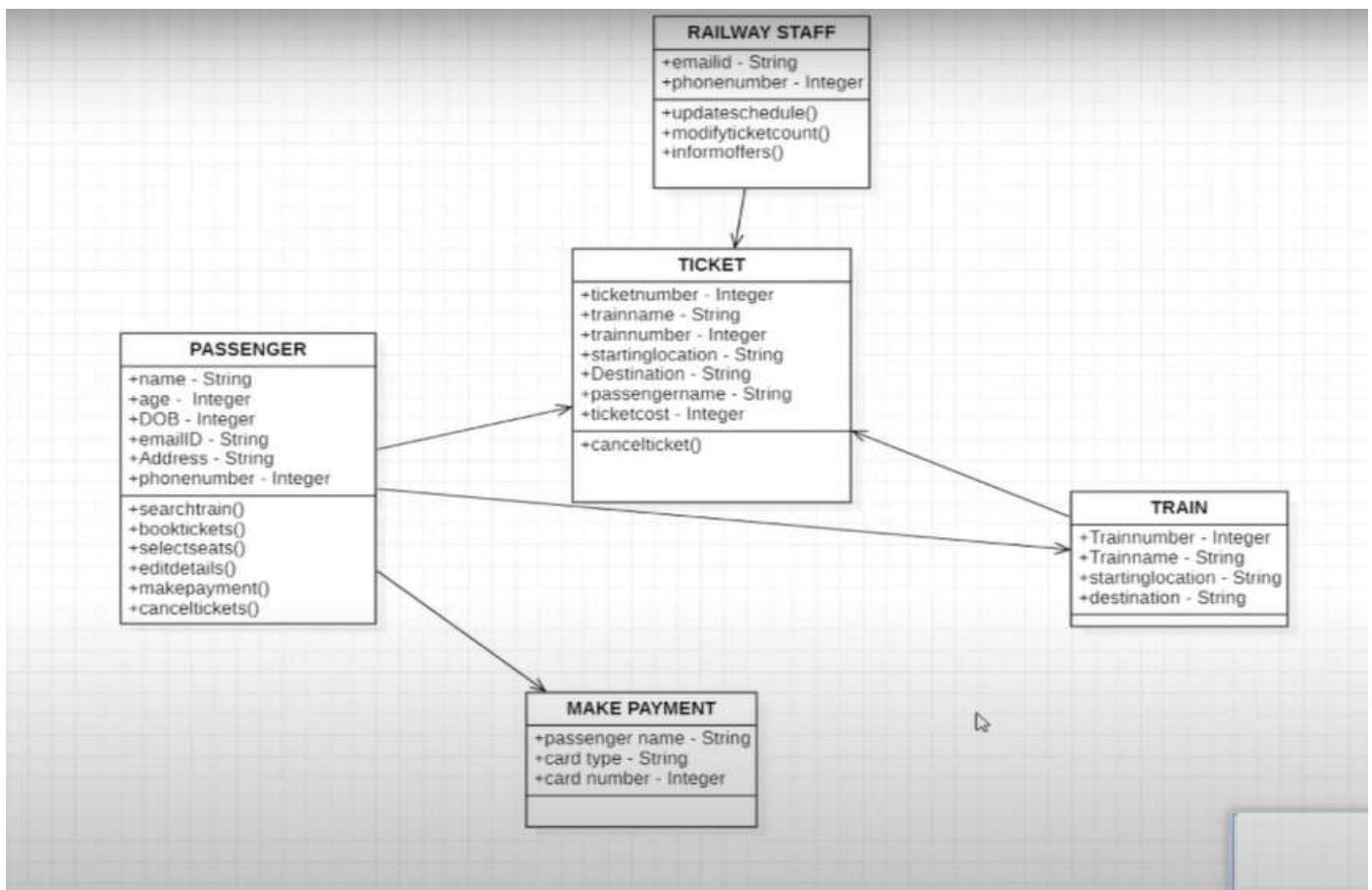
**Aggregation:** An aggregation is a subset of association, which represents has a relationship. It is more specific than association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class. The company encompasses a number of employees, and even if one employee resigns, the company still exists.



**Composition:** The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship. A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost.



**Example:** Class diagram for railway reservation system

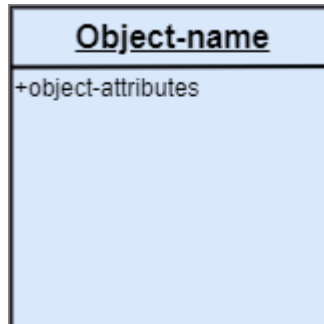


## OBJECT DIAGRAM

- An **Object Diagram** describes the instance of a class. It visualizes the particular functionality of a system. The difference between the class and object diagram is that the class diagram mainly represents the bird's eye view of a system which is also referred to as an abstract view.

- Objects are the real-world entities whose behavior is defined by the classes. Objects are used to represent the static view of an object-oriented system. We cannot define an object without its class. Object and class diagrams are somewhat similar.

### Notation of an Object Diagram



### How to Draw Object Diagram

Below are the steps to draw Object Diagram in UML:

**Step 1)** Before drawing an object diagram, one should analyze all the objects inside the system.

**Step 2)** The relations of the object must be known before creating the diagram.

**Step 3)** Association between various objects must be cleared before.

**Step 4)** An object should have a meaningful name that describes its functionality.

**Step 5)** An object must be explored to analyze various functionalities of it.

### Purpose of Object Diagram

- It is used to describe the static aspect of a system.
- It is used to represent an instance of a class.
- It can be used to perform forward and reverse engineering on systems.
- It is used to understand the behavior of an object.
- It can be used to explore the relations of an object and can be used to analyze other connecting objects.

### Applications of Object diagrams

The following are the application areas where the object diagrams can be used.

1. To build a prototype of a system.
2. To model complex data structures.
3. To perceive the system from a practical perspective.
4. Reverse engineering.

## DIFFERENCE BETWEEN CLASS AND OBJECT

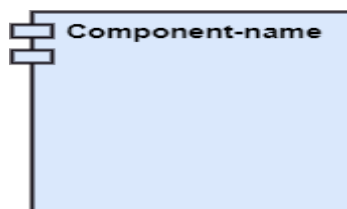
S No.	Class Diagram	Object Diagram
1.	It depicts the static view of a system.	It portrays the real-time behavior of a system.
2.	Dynamic changes are not included in the class diagram.	Dynamic changes are captured in the object diagram.
3.	The data values and attributes of an instance are not involved here.	It incorporates data values and attributes of an entity.
4.	The object behavior is manipulated in the class diagram.	Objects are the instances of a class.

## COMPONENT DIAGRAM

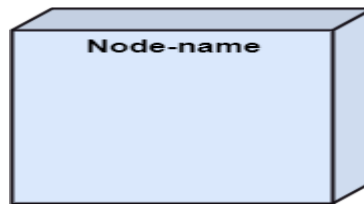
- A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.
- It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system.
- A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behavior is explained by the provided and required interfaces.

### Notation of a Component Diagram

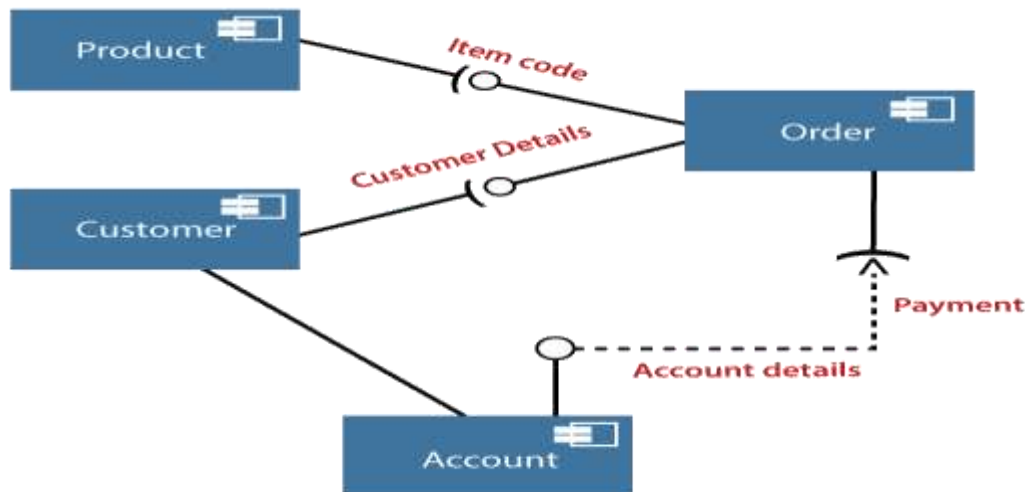
a) A component



b) A node



### Example of a Component Diagram



### Where to use Component Diagrams?

The component diagram is a special purpose diagram, which is used to visualize the static implementation view of a system. It represents the physical components of a system, or we can say it portrays the organization of the components inside a system. The components, such as libraries, files, executables, etc. are first needed to be organized before the implementation.

The component diagram can be used for the followings:

1. To model the components of the system.
2. To model the schemas of a database.
3. To model the applications of an application.
4. To model the system's source code.

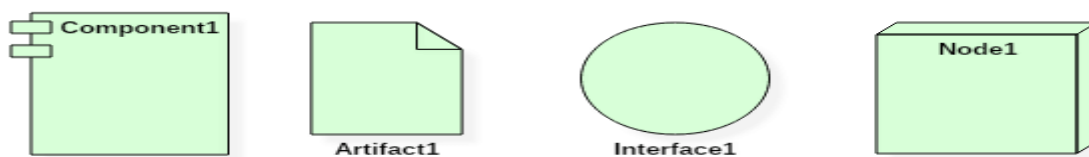
### Benefits of component diagrams

- Imagine the system's physical structure.
- Pay attention to the system's components and how they relate.
- Emphasize the service behavior as it relates to the interface.

## DEPLOYMENT DIAGRAM

- A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them.
- Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware.
- Deployment diagrams help model the hardware topology of a system compared to other UML diagram types which mostly outline the logical components of a system.

### Deployment Diagram Symbol and notations



**Deployment Diagram Notations**

A deployment diagram consists of the following notations:

1. A node
2. A component
3. An artifact
4. An interface

### What is an artifact?

An artifact represents the specification of a concrete real-world entity related to software development. You can use the artifact to describe a framework which is used during the software development process or an executable file. Artifacts are deployed on the nodes. The most common artifacts are as follows,

1. Source files
2. Executable files
3. Database tables
4. Scripts
5. DLL files
6. User manuals or documentation
7. Output files



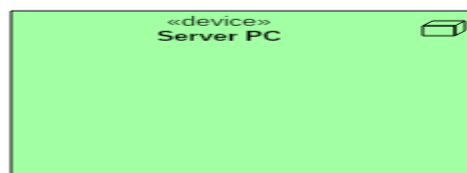
**Artifact Instances:** An artifact instance represents an instance of a particular artifact. An artifact instance is denoted with same symbol as that of the artifact except that the name is underlined. UML diagram allows this to differentiate between the original artifact and the instance. Each physical copy or a file is an instance of a unique artifact.

Generally, an artifact instance is represented as follows in the unified modeling language.

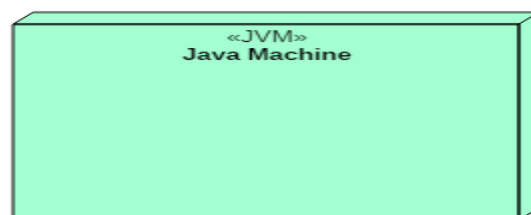


### What is a node?

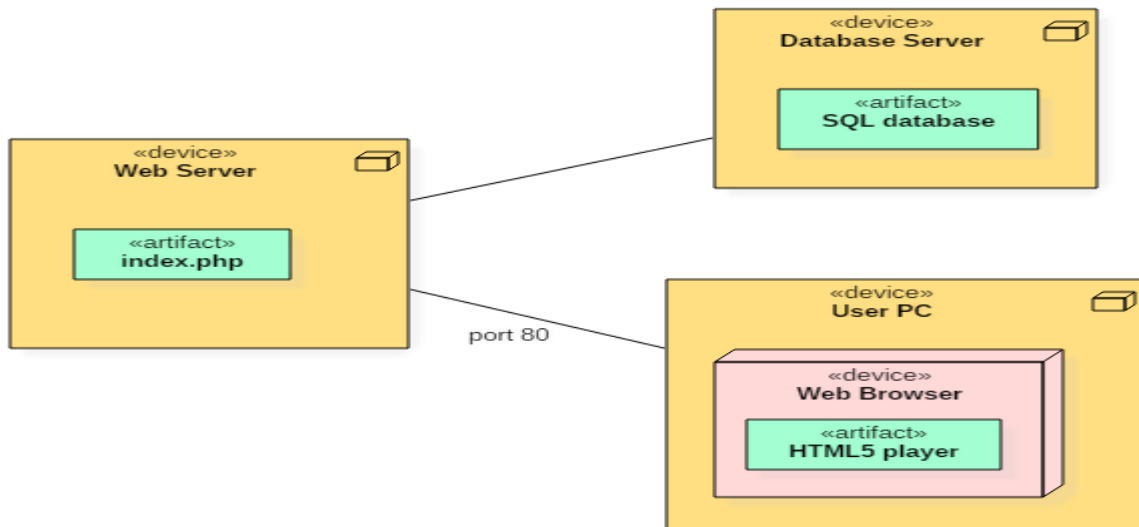
Node is a computational resource upon which artifacts are deployed for execution. A node is a physical thing that can execute one or more artifacts. A node may vary in its size depending upon the size of the project. Node is an essential UML element that describes the execution of code and the communication between various entities of a system. It is denoted by a 3D box with the node-name written inside of it. Nodes help to convey the hardware which is used to deploy the software.



**<< execution environment >>:** It is a node that represents an environment in which software is going to execute. For example, Java applications are executed in java virtual machine (JVM). JVM is considered as an execution environment for Java applications. We can nest an execution environment into a device node. You can nest more than one execution environments in a single device node. Following is a representation of an execution environment in UML:



### Example:



## BEHAVIOURAL DIAGRAMS

Behavioral diagrams portray a dynamic view of a system or the behavior of a system, which describes the functioning of the system. It includes use case diagrams, state diagrams, and activity diagrams. It defines the interaction within the system.





- **State Machine Diagram:** It is a behavioral diagram. it portrays the system's behavior utilizing finite state transitions. It is also known as the **State-charts** diagram. It models the dynamic behavior of a class in response to external stimuli.
- **Activity Diagram:** It models the flow of control from one activity to the other. With the help of an activity diagram, we can model sequential and concurrent activities. It visually depicts the workflow as well as what causes an event to occur.
- **Use Case Diagram:** It represents the functionality of a system by utilizing actors and use cases. It encapsulates the functional requirement of a system and its association with actors. It portrays the use case view of a system.
- **Sequence Diagram:** Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.



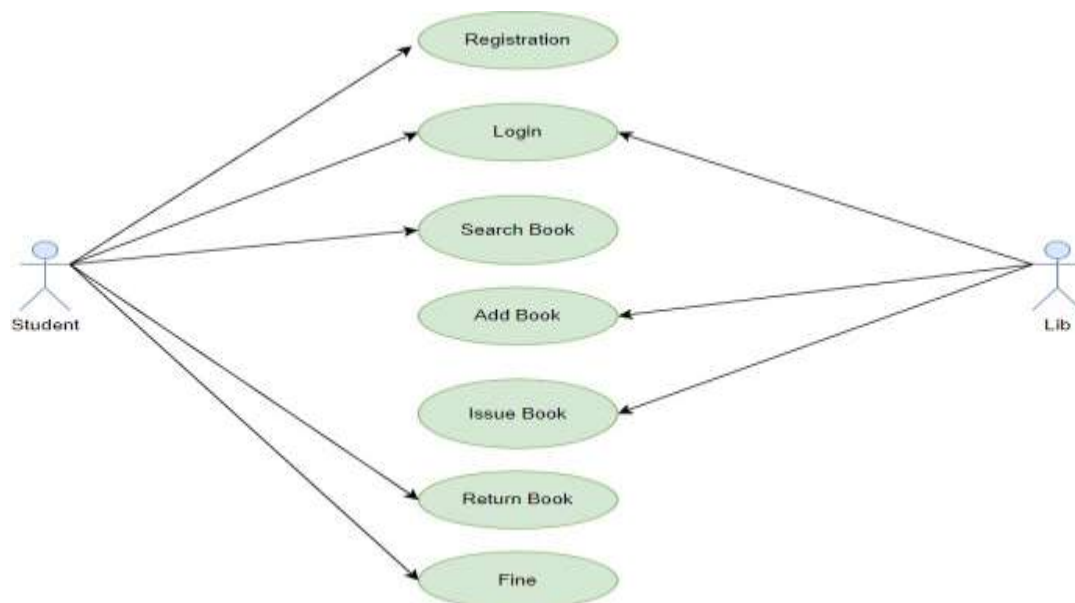
## USE CASE DIAGRAM

- A use case is a description of a set of interactions between a user and the system. Use case diagrams referred as a Behavior model or diagram. It simply describes and displays the relation or interaction between the users or customers and providers of application service or the system.
- It describes different actions that a system performs in collaboration to achieve something with one or more users of the system. Use case diagram is used a lot nowadays to manage the system.

### Components of use case diagram

Notation	Meaning
	Use case: An Ellipse represents a single Use Case. The name of the use case is written inside ellipse.
	Actor: Represents every element which interact with the system. It may be a user a another system which interact with the system described in the use case.
	Ordinary Relationship: Represents a connection that is a channel for the transfer of information between a use case and an actor or between use cases.
	System Boundary: A square represents the boundary of the system. Inside the system are the use cases and outside the square are the actors who interact with the system.

### EXAMPLE USECASE DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM



## **Important**

Following are some important tips that are to be kept in mind while drawing a use case diagram:

- A simple and complete use case diagram should be articulated.
- A use case diagram should represent the most significant interaction among the multiple interactions.
- At least one module of a system should be represented by the use case diagram.
- If the use case diagram is large and more complex, then it should be drawn more generalized.

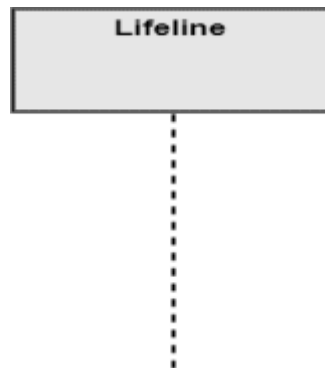
## **SEQUENCE DIAGRAM**

- Sequence diagram is a dynamic model of a use case, showing the interaction among classes during a specified time period.
- Sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur.
- Describe the flow of messages, events, actions between objects
- Show concurrent processes and activations
- Show time sequences that are not easily depicted in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

## **Notations of a Sequence**

### **Diagram Lifeline:**

An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.



**Messages** The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.

Following are types of messages enlisted below:

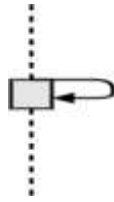
**Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.



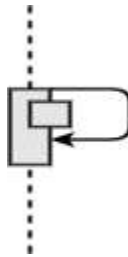
**Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.



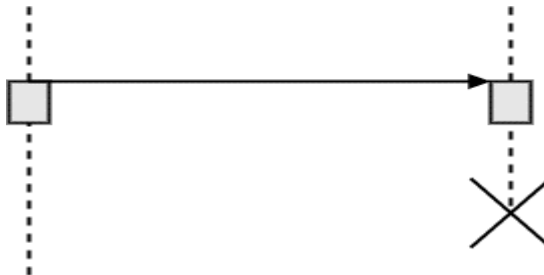
**Self-Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



**Recursive Message:** A self-message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self-message as it represents the recursive calls.

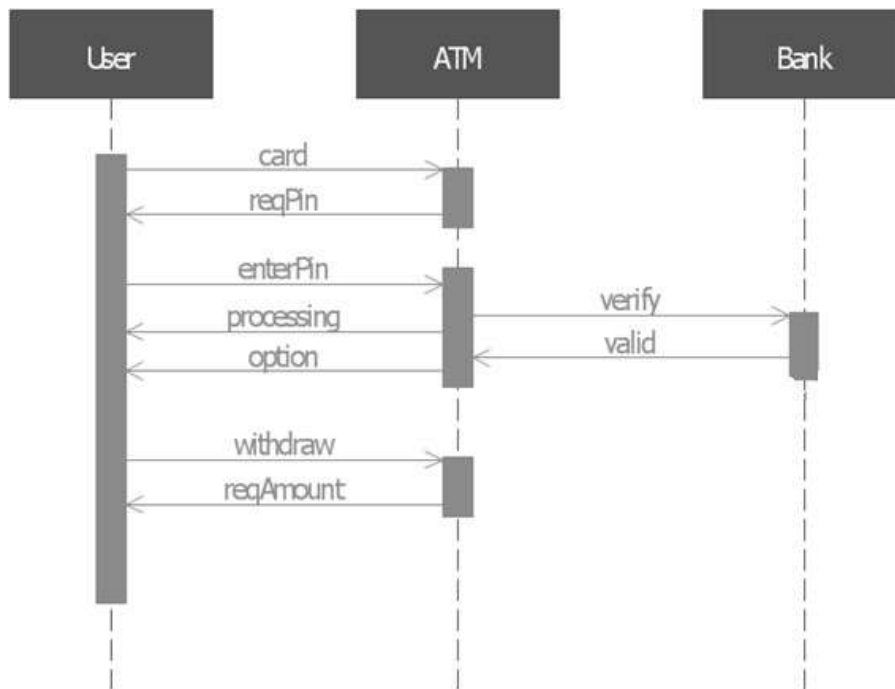


**Destroy Message:** It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.



### EXAMPLE SEQUENCE DIAGRAM FOR ATM SYSTEM

Sequence diagrams are great at showing the interactions between elements in a system over time. An example might be how a banking client interacts with an ATM to withdraw cash.



## COLLABORATION DIAGRAM

- A Collaboration is a collection of named objects and actors with links connecting them. They collaborate in performing some task.
- A Collaboration defines a set of participants and relationships that are meaningful for a given set of purposes
- A Collaboration between objects working together provides emergent desirable functionalities in Object-Oriented systems
- Each object (responsibility) partially supports emergent functionalities
- Objects are able to produce (usable) high-level functionalities by working together
- Objects collaborate by communicating (passing messages) with one another in order to work together

### Notations of Collaboration Diagram

**Objects:** An object is represented by an object symbol showing the name of the object and its class underlined, separated by a colon:

Object name : class name

You can use objects in collaboration diagrams in the following ways:

M.Srikanth,  
M.Tech., (Ph.D)  
Asst.Prof, Dept of IT, VIT

Each object in the collaboration is named and has its class specified

- Not all classes need to appear
- There may be more than one object of a class
- An object's class can be unspecified. Normally you create a collaboration diagram with objects first and specify their classes later.
- The objects can be unnamed, but you should name them if you want to discriminate different objects of the same class.

**Actors:** Normally an actor instance occurs in the collaboration diagram, as the invoker of the interaction. If you have several actor instances in the same diagram, try keeping them in the periphery of the diagram.

- Each Actor is named and has a role
- One actor will be the initiator of the use case

**Links:** Links connect objects and actors and are instances of associations and each link corresponds to an association in the class diagram

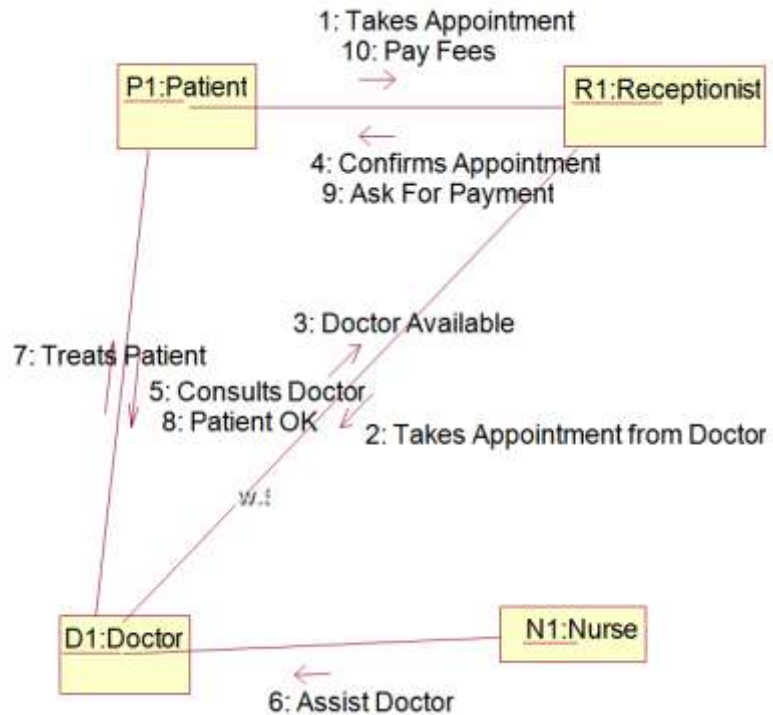
Links are defined as follows:

- A link is a relationship among objects across which messages can be sent. In collaboration diagrams, a link is shown as a solid line between two objects.
- An object interacts with, or navigates to, other objects through its links to these objects.
- A link can be an instance of an association, or it can be anonymous, meaning that its association is unspecified.
- Message flows are attached to links, see Messages.

**Messages:** A message is a communication between objects that conveys information with the expectation that activity will ensue. In collaboration diagrams, a message is shown as a labeled arrow placed near a link.

- The message is directed from sender to receiver
- The receiver must understand the message
- The association must be navigable in that direction

### Example:



## STATE CHART DIAGRAM

State chart diagrams provide us an efficient way to model the interactions or communication that occur within the external entities and a system. These diagrams are used to model the event-based system. A state of an object is controlled with the help of an event. State chart diagrams are used to describe various states of an entity within the application system.

There are a total of two types of state machine diagram in UML:

### 1. Behavioral State Machine Diagram

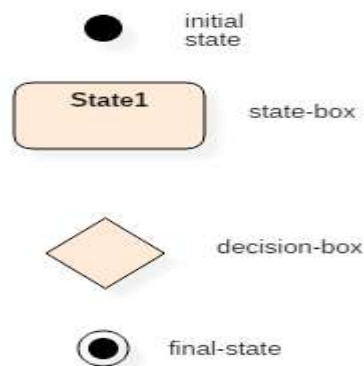
- It captures the behavior of an entity present in the system.
- It is used to represent the specific implementation of an element.
- The behavior of a system can be modeled using behavioral state machine diagram in OOAD.

## 2. Protocol State Machine Diagram

- These diagrams are used to capture the behavior of a protocol.
- It represents how the state of protocol changes concerning the event. It also represents corresponding changes in the system.
- They do not represent the specific implementation of an element.

### Notation and Symbol for State Machine Diagram (Statechart Diagram)

Following are the various notations that are used throughout the state chart diagram. All these notations, when combined, make up a single diagram.



UML state diagram notations

**Initial state:** The initial state symbol is used to indicate the beginning of a state machine diagram.

**Final state:** This symbol is used to indicate the end of a state machine diagram.

**Decision box:** It contains a condition. Depending upon the result of an evaluated guard condition, a new path is taken for program execution.

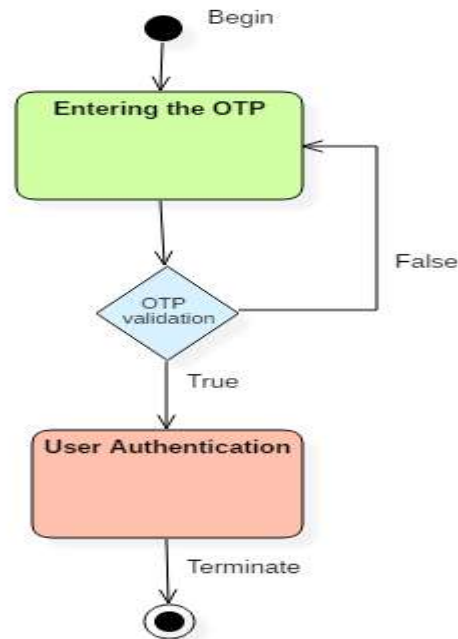
**Transition:** A transition is a change in one state into another state which is occurred because of some event. A transition causes a change in the state of an object.

**State box:** It is a specific moment in the lifespan of an object. It is defined using some condition or a statement within the classifier body. It is used to represent any static as well as dynamic situations.



## Example of State Machine

Following state diagram example chart represents the user authentication process.



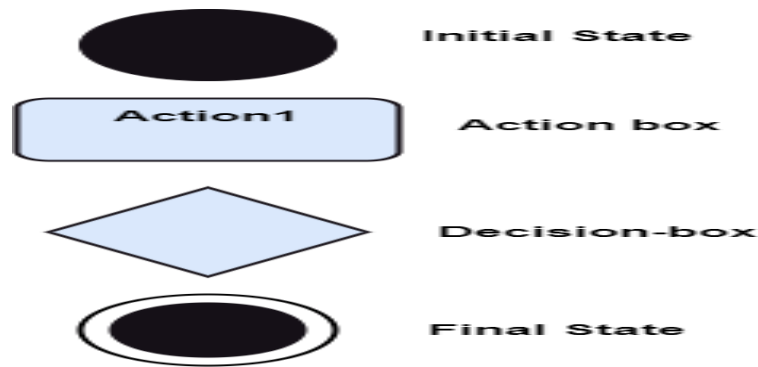
## ACTIVITY DIAGRAM

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning (an initial state) and an end (a final state).

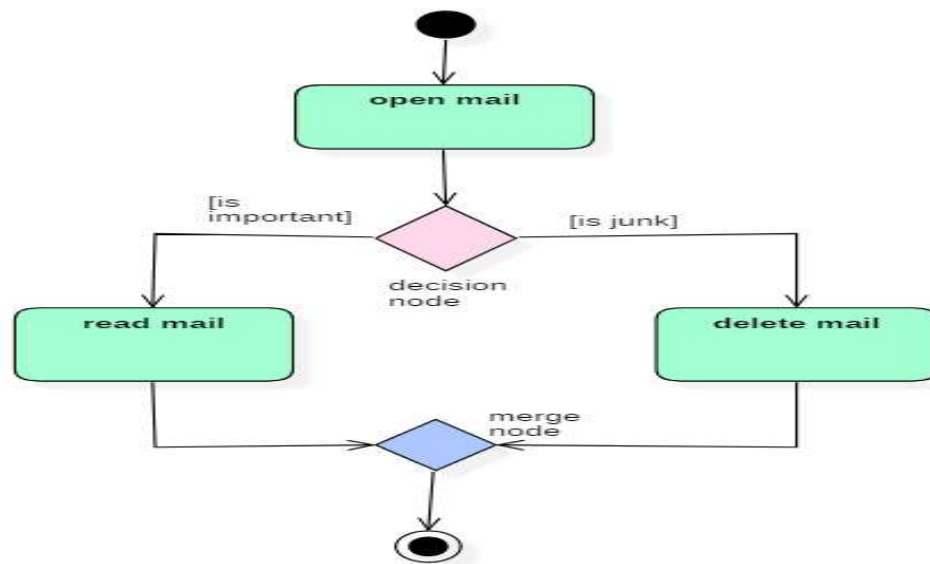
### Notation of an Activity diagram

Activity diagram constitutes following notations:

- **Initial State:** It depicts the initial stage or beginning of the set of actions.
- **Final State:** It is the stage where all the control flows and object flows end.
- **Decision Box:** It makes sure that the control flow or object flow will follow only one path.
- **Action Box:** It represents the set of actions that are to be performed.



**Example:**



### When to use an Activity Diagram?

1. To graphically model the workflow in an easier and understandable way.
2. To model the execution flow among several activities.
3. To model comprehensive information of a function or an algorithm employed within the system.
4. To model the business process and its workflow.
5. To envision the dynamic aspect of a system.
6. To generate the top-level flowcharts for representing the workflow of an application.
7. To represent a high-level view of a distributed or an object-oriented system.

## UNIT – IV

### CODING AND TESTING

#### CODING

The coding is the process of transforming the design of a system into a computer language format. This coding phase of software development is concerned with software translating design specification into the source code. It is necessary to write source code & internal documentation so that conformance of the code to its specification can be easily verified.

Coding is done by the coder or programmers who are independent people than the designer. The goal is not to reduce the effort and cost of the coding phase, but to cut to the cost of a later stage. The cost of testing and maintenance can be significantly reduced with efficient coding.

#### Coding Standards

General coding standards refers to how the developer writes code, so here we will discuss some essential standards regardless of the programming language being used.

**The following are some representative coding standards:**

#### Coding Standards



1. **Indentation:** Proper and consistent indentation is essential in producing easy to read and maintainable programs.

Indentation should be used to:

- Emphasize the body of a control structure such as a loop or a select statement.
- Emphasize the body of a conditional statement
- Emphasize a new scope block

M.Srikanth

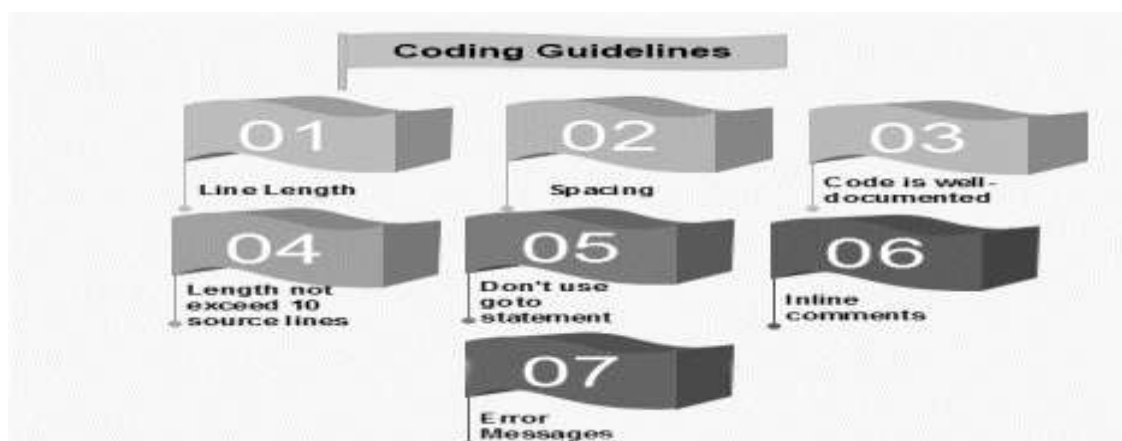
M.Tech.,(Ph.D)

Asst.Prof, Dept of IT, VIT

2. **Inline comments:** Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.
3. **Rules for limiting the use of global:** These rules file what types of data can be declared global and what cannot.
4. **Structured Programming:** Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain, except as outlined line in the FORTRAN Standards and Guidelines.
5. **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
6. **Error return conventions and exception handling system:** Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently.

## Coding Guidelines

- General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain. Most of the examples use the C language syntax, but the guidelines can be tested to all languages.
- The following are some representative coding guidelines recommended by many software development organizations.



M.Srikanth

M.Tech.,(Ph.D)

Asst.Prof, Dept of IT, VIT

**1. Line Length:** It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

**2. Spacing:** The appropriate use of spaces within a line of code can improve readability.

**Example:**

**Bad:**     `cost=price+(price*sales_tax)`  
          `fprintf(stdout,"The total cost is %5.2f\n",cost);`

**Better:**   `cost = price + ( price * sales_tax )`  
          `fprintf (stdout,"The total cost is %5.2f\n",cost);`

**3. The code should be well-documented:** As a rule of thumb, there must be at least one comment line on the average for every three-source line.

**4. The length of any function should not exceed 10 source lines:** A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.

**5. Do not use goto statements:** Use of goto statements makes a program unstructured and very tough to understand.

**6. Inline Comments:** Inline comments promote readability.

**7. Error Messages:** Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

## CODE REVIEW

Code Review is a systematic examination, which can find and remove the vulnerabilities in the code such as memory leaks and buffer overflows.

- Technical reviews are well documented and use a well-defined defect detection process that includes peers and technical experts.
- It is ideally led by a trained moderator, who is NOT the author.

M.Srikanth  
M.Tech.,(Ph.D)  
Asst.Prof, Dept of IT, VIT

- This kind of review is usually performed as a peer review without management participation.
- Reviewers prepare for the review meeting and prepare a review report with a list of findings.
- Technical reviews may be quite informal or very formal and can have a number of purposes but not limited to discussion, decision making, evaluation of alternatives, finding defects and solving technical problems.



### Types of code reviews:

**Personal review:** code author reviews his/her own work

**Peer review:** Review by another team member

**Inspection:** Review by whole team

## SOFTWARE DOCUMENTATION

**Software documentation** is a written piece of text that is often accompanied with a software program. This makes the life of all the members associated with the project more easy. It may contain anything from API documentation, build notes or just help content. It is a very critical process in software development. It's primarily an integral part of any computer code development method. Moreover, the computer code practitioners are a unit typically concerned with the worth, degree of usage and quality of the actual documentation throughout development and its maintenance throughout the total method. Motivated by the requirements of Novatel opposition, a world leading company developing package in support of worldwide navigation satellite system, and based mostly on the results of a former systematic mapping studies area unit aimed of the higher understanding of the usage and therefore the quality of a varied technical documents throughout computer code development and their maintenance.

For example, before development of any software product requirements are documented which is called as Software Requirement Specification (SRS). Requirement gathering is considered as an stage of Software Development Life Cycle (SDLC).

M.Srikanth

M.Tech.,(Ph.D)

Asst.Prof, Dept of IT, VIT

## Types Of Software Documentation:

### 1. Requirement Documentation:

It is the description of how the software shall perform and which environment setup would be appropriate to have the best out of it. These are generated while the software is under development and is supplied to the tester groups too.

### 2. Architectural Documentation:

Architecture documentation is a special type of documentation that concerns the design. It contains very little code and is more focused on the components of the system, their roles and working. It also shows the data flows throughout the system.

### 3. Technical Documentation:

These contain the technical aspects of the software like API, algorithms etc. It is prepared mostly for the software develop.

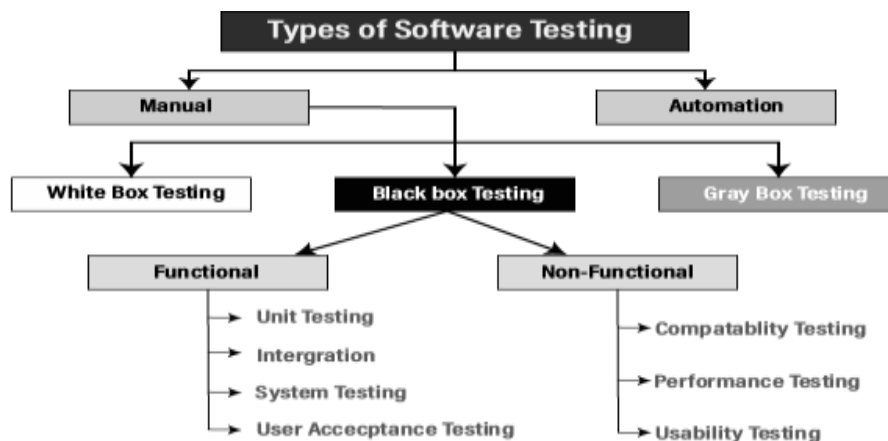
### 4. End-user Documentation:

As the name suggests these are made for the end user. It contains support resources for the end user.

## TESTING

Testing means software testing. It finding the bug/error in the developed software or existing software. Software testing is a process used to identify the correctness, completeness and quality of developed software. It includes set of activities finding errors.

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.



## UNIT TESTING

- It is also called component testing
- It is performed on standalone module to check whether it is developed correctly
- Unit testing, a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules. The main aim is to isolate each unit of the system to identify, analyse and fix the defects.



### Unit Testing Techniques

The Unit Testing Techniques are mainly categorized into three parts which are Black box testing that involves testing of user interface along with input and output, White box testing that involves testing the functional behaviour of the software application and Gray box testing that is used to execute test suites, test methods, test cases and performing risk analysis.

Code coverage techniques used in Unit Testing are listed below:

- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage
- Finite State Machine Coverage

### Advantage

- Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.
- Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (i.e. Regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.



- Due to the modular nature of the unit testing, we can test parts of the project without waiting for others to be completed.

### Disadvantages

- Unit testing can't be expected to catch every error in a program. It is not possible to evaluate all execution paths even in the most trivial programs
- Unit testing by its very nature focuses on a unit of code. Hence it can't catch integration errors or broad system level errors.

## BLACK-BOX TESTING

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

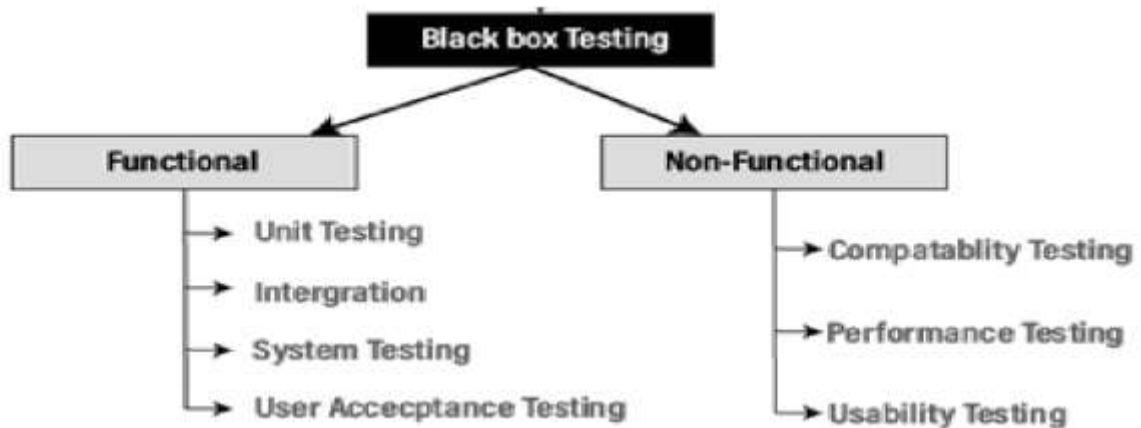
In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.



### 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.



### Functional Testing

It is a type of software testing which is used to verify the functionality of the software application, whether the function is working according to the requirement specification. In functional testing, each function tested by giving the value, determining the output, and verifying the actual output with the expected value. Functional testing performed as black-box testing which is presented to confirm that the functionality of an application or system behaves as we are expecting. It is done to verify the functionality of the application.

Functional testing also called as black-box testing, because it focuses on application specification rather than actual code. Tester has to test only the program rather than the system.

#### Types of Functional Testing

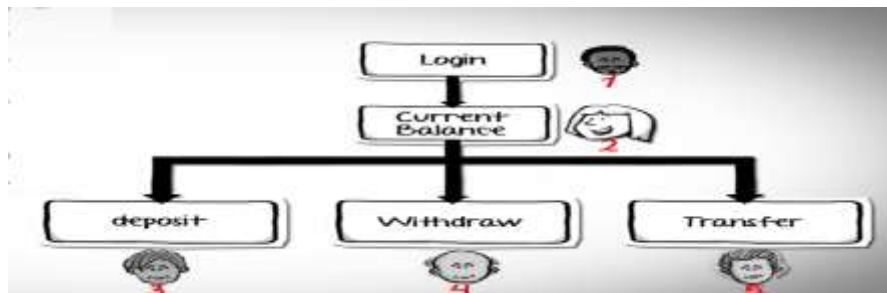
1. Unit Testing
2. Integrity Testing
3. System Testing
4. Acceptance Testing

#### Unit Testing

- It is also called component testing
- It is performed on standalone module to check whether it is developed correctly
- Unit testing, a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules. The main aim is to isolate each unit of the system to identify, analyse and fix the defects.

## Integration Testing

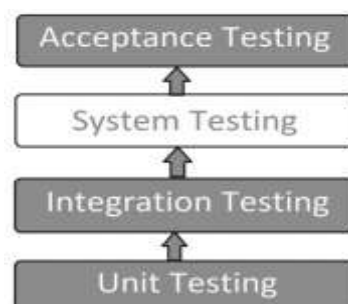
- In this phase of testing, individual modules are combined and tested as a group
- Data transfer between the modules is tested thoroughly
- Integration testing is carried out by testers
- Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updating etc.



## 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

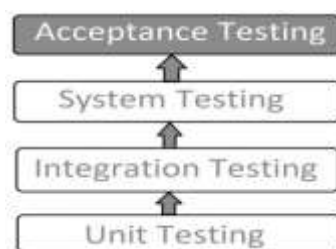


## Types of System Testing:

1. **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
2. **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
3. **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
4. **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

## Acceptance Testing

It is a formal testing according to user needs, requirements and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users, customers or other authorized entities to determine whether to accept the system or not.



## Two ways of testing

- **Alpha:** A small set of employees of the client
- **Beta:** A small set of customers

## Use of Acceptance Testing:

- To find the defects missed during the functional testing phase.
- How well the product is developed.
- A product is what actually the customers need.
- Feedbacks help in improving the product performance and user experience.
- Minimize or eliminate the issues arising from the production.

## Non-Functional Testing

- Non-Functional Testing is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc) of a software application. It is designed to test the readiness of a system as per non-functional parameters which are never addressed by functional testing.
- An excellent example of non-functional test would be to check how many people can simultaneously login into software.
- Non-functional testing is equally important as functional testing and affects client satisfaction.

### Types of Non-Functional Testing

1. Compatibility Testing
2. Performance Testing
3. Usability Testing

#### 1. Compatibility Testing

- Compatibility Testing is a type of Software testing to check whether your software is capable of running on different hardware, operating systems, applications, network environments or Mobile devices.
- Compatibility Testing is a type of Non-functional testing
- Software should be installable on all versions of Windows and Mac



#### 2. Performance Testing

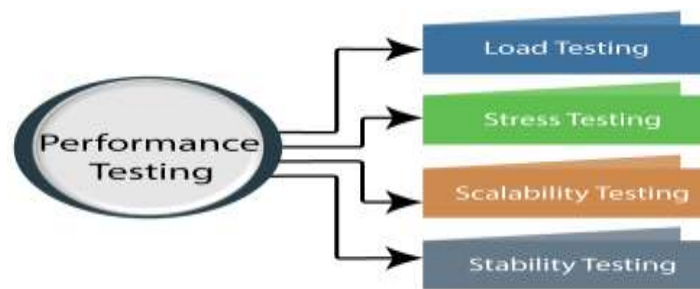
- It is the most important part of non-functional testing.
- Checking the behavior of an application by applying some load is known as performance testing
- Application load time should not be more than 5 secs up to 1000 users accessing it simultaneously

M.Srikanth

M.Tech.,(Ph.D)

Asst.Prof, Dept of IT, VIT

## Types of Performance Testing



**Load testing:** The load testing is used to check the performance of an application by applying some load which is either less than or equal to the desired load is known as load testing. **For example:** In the below image, **1000 users** are the **desired load**, which is given by the customer, and **3/second** is the **goal** which we want to achieve while performing a load testing.



**Stress Testing:** The stress testing is testing, which checks the behavior of an application by applying load greater than the desired load. **For example:** If we took the above example and increased the desired load 1000 to 1100 users, and the goal is 4/second. While performing the stress testing in this scenario, it will pass because the load is greater (100 up) than the actual desired load.



**Scalability Testing:** Checking the performance of an application by increasing or decreasing the load in particular scales (no of a user) is known as scalability testing. Upward scalability and downward scalability testing are called scalability testing.

**Stability Testing:** Checking the performance of an application by applying the load for a particular duration of time is known as Stability Testing.

### 3. Usability Testing

- Usability Testing is a significant type of software testing technique, which is comes under the non-functional testing.
- Checking the user-friendliness, efficiency, and accuracy of the application is known as Usability Testing.

## 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, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.



### Working process

- **Input:** Requirements, Functional specifications, design documents, source code.
- **Processing:** Performing risk analysis for guiding through the entire process.
- **Proper test planning:** Designing test cases so as to cover entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
- **Output:** Preparing final report of the entire testing process.

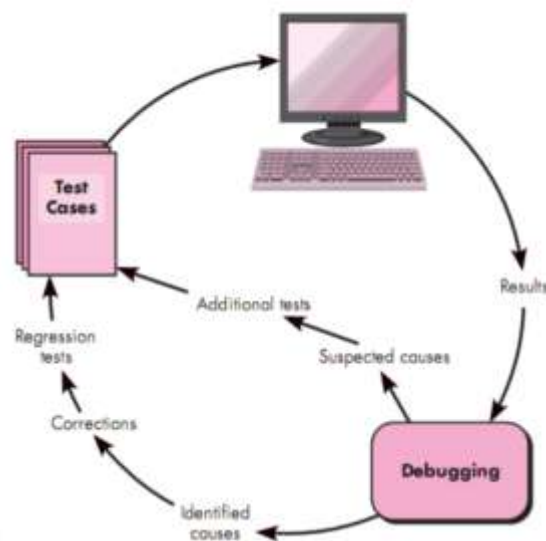
### White Box Testing Techniques:

- **Statement Coverage** - This technique is aimed at exercising all programming statements with minimal tests.
- **Branch Coverage** - This technique is running a series of tests to ensure that all branches are tested at least once.
- **Path Coverage** - This technique corresponds to testing all possible paths which means that each statement and branch is covered.

## DEBUGGING

It is a systematic process of spotting and fixing the number of bugs, or defects, in a piece of software so that the software is behaving as expected. Debugging is harder for complex systems in particular when various subsystems are tightly coupled as changes in one system or interface may cause bugs to emerge in another.

Debugging is a developer activity and effective debugging is very important before testing begins to increase the quality of the system. Debugging will not give confidence that the system meets its requirements completely but testing gives confidence.



### Debugging Process:

- Problem identification and report preparation.
- Assigning the report to software engineer to the defect to verify that it is genuine.
- Defect Analysis using modeling, documentations, finding and testing candidate flaws, etc.
- Defect Resolution by making required changes to the system.
- Validation of corrections.

## PROGRAM ANALYSIS TOOL

**Program Analysis Tool** is an automated tool whose input is the source code or the executable code of a program and the output is the observation of characteristics of the program. It gives various characteristics of the program such as its size, complexity, adequacy of commenting, adherence to programming standards and many other characteristics.

M.Srikanth

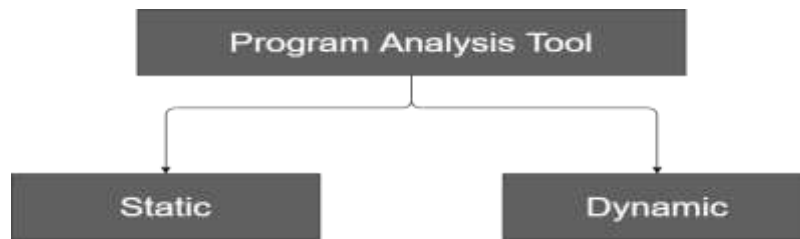
M.Tech.,(Ph.D)

Asst.Prof, Dept of IT, VIT



## Classification of Program Analysis Tools:

Program Analysis Tools are classified into two categories:



### 1. Static Program Analysis Tools:

Static Program Analysis Tool is such a program analysis tool that evaluates and computes various characteristics of a software product without executing it. Normally, static program analysis tools analyze some structural representation of a program to reach a certain analytical conclusion. Basically some structural properties are analyzed using static program analysis tools.

The structural properties that are usually analyzed are:

1. Whether the coding standards have been fulfilled or not.
2. Some programming errors such as uninitialized variables.
3. Mismatch between actual and formal parameters.
4. Variables that are declared but never used.

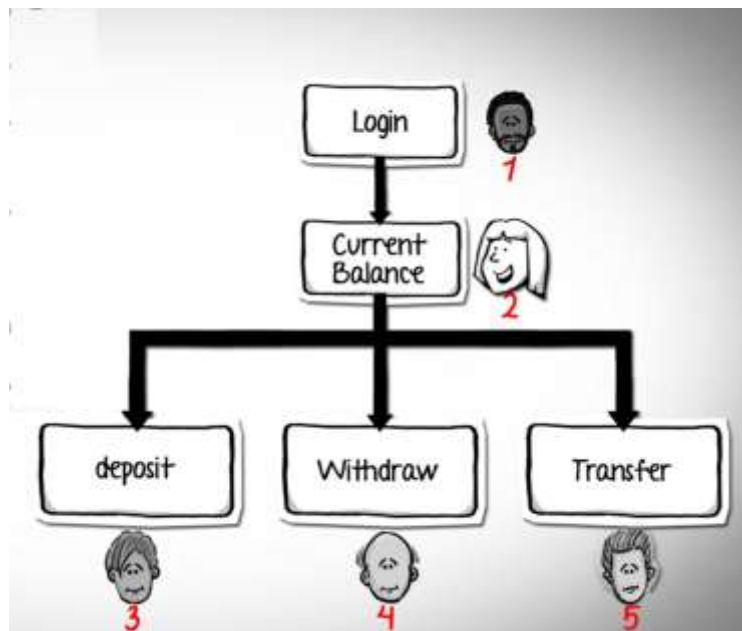
Code walkthroughs and code inspections are considered as static analysis methods but static program analysis tool is used to designate automated analysis tools. Hence, a compiler can be considered as a static program analysis tool.

### 2. Dynamic Program Analysis Tools:

Dynamic Program Analysis Tool is such type of program analysis tool that require the program to be executed and its actual behavior to be observed. A dynamic program analyzer basically implements the code. Once the software is tested and its behavior is observed, the dynamic program analysis tool performs a post execution analysis and produces reports which describe the structural coverage that has been achieved by the complete testing process for the program. The output of a dynamic program analysis tool can be stored and printed easily and provides evidence that complete testing has been done. The result of dynamic analysis is the extent of testing performed as white box testing.

## INTEGRATION TESTING

- In this phase of testing, individual modules are combined and tested as a group
- Data transfer between the modules is tested thoroughly
- Integration testing is carried out by testers
- Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updating etc.



### Types of integration testing

1. **Big Bang:** In Big Bang integration testing, all the components are integrated and tested together in one go.
2. **Top Down:** Top down integration testing involves testing top-level units first and then moving on to testing lower-level units.
3. **Bottom up:** Bottom up integration testing involves testing the lower level modules first. These tests are then used to test higher level modules until all the modules are tested.
4. **Sandwich testing:** Sandwich integration testing is a combination of the Top Down and Bottom Up approaches.

## TESTING OBJECT-ORIENTED PROGRAMS

The object-oriented paradigm is gaining popularity because of its benefits in analysis, design, and coding. Conventional testing methods cannot be applied for testing classes because of inheritance, dynamic binding, message, passing, polymorphism, concurrency, etc. Classes are a fundamentally different problem from testing functions.

According to Davis the dependencies occurring in conventional systems are:

- Data dependencies between variables
- Calling dependencies between modules
- Functional dependencies between a module and the variable it computes
- Definitional dependencies between a variable and its types.

But in Object-Oriented systems there are following additional dependencies:

- Class to class dependencies
- Class to method dependencies
- Class to message dependencies
- Class to variable dependencies
- Method to variable dependencies
- Method to message dependencies
- Method to method dependencies

### Issues in Testing Classes:

- In object-oriented programs, control flow is characterized by message passing among objects, and the control flow switches from one object to another by inter-object communication.
- This lack of sequential control flow within a class requires different approaches for testing. The concept of inheritance opens various issues. e.g., if changes are made to a parent class or super class, it will be difficult to test subclasses individually.

Techniques of object-oriented testing are as follows:

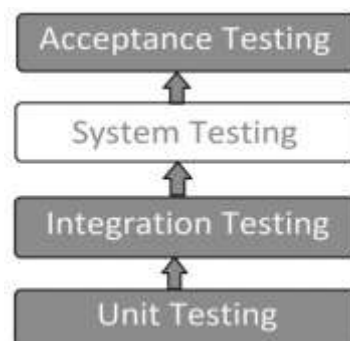
1. **Fault Based Testing:** This type of checking permits for coming up with test cases supported the consumer specification or the code or both. It tries to identify possible faults (areas of design or code that may lead to errors) In the object-oriented model, interaction errors can be uncovered by scenario-based testing.

2. **Class Testing Based on Method Testing:** This approach is the simplest approach to test classes. Each method of the class performs a well defined cohesive function and can, therefore, be related to unit testing of the traditional testing techniques. Therefore all the methods of a class can be involved at least once to test the class.
3. **Random Testing:** It is supported by developing a random test sequence that tries the minimum variety of operations typical to the behavior of the categories
4. **Partition Testing:** This methodology categorizes the inputs and outputs of a category so as to check them severely. This minimizes the number of cases that have to be designed.
5. **Scenario-based Testing:** It primarily involves capturing the user actions then stimulating them to similar actions throughout the test. These tests tend to search out interaction form of error.

## 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



### Types of System Testing:

1. **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
2. **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.

M.Srikanth

M.Tech.,(Ph.D)

Asst.Prof, Dept of IT, VIT

3. **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
4. **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

### **SOME GENERAL ISSUES ASSOCIATED WITH TESTING.**

The pressure to bring high-quality products into the market means the product must undergo several rounds of testing to ensure issues are identified and rectified in time. But the process of developing test cases and carrying out manual or automated testing is not as easy as it appears; testers often come across many challenges along the testing lifecycle that makes the entire process rather taxing.

**1. Undefined Quality Standards:** Undefined or poorly defined quality standards means there is no clarity on testing requirements, specifications, guidelines, or characteristics. In the absence of the right standards, it becomes extremely difficult for testers to satisfy customer needs and meet quality requirements, as well as comply with regulations.

**2. Test Environment Duplication:** Using a real-life environment to test software products is important. This ensures the product is tested across all possible configuration combinations. A test environment that is as production-like as possible also reduces the likelihood of introducing new features that may not be compatible with the production environment.

**3. Lack of Communication:** Testers tend to work in complete isolation, and rarely indulge in communicating with other team members. Such communication often leads to testers spending more in back-and-forth emails, attending meetings, taking phone calls, and providing status updates. Ensuring great communication across the team means testers have access to all the information they need to start testing quickly and efficiently.

**4. Unstable Environment:** Software developers face a major challenge with respect to the presence of unstable test environments. Unstable environments tend to potentially disrupt the overall release process. To overcome issues with unstable environments, you need to get into the practice of formalizing test environment requirements early in the testing lifecycle.

**5. Insufficient Requirements Gathering:** Too many missed requirements exert immense pressure on project schedules and compel testers to skip test cases to save time. Proper requirements gathering ensures teams know what features the product is supposed to offer, the level of functionality expected as well as in unearthing the right defects and issues.

M.Srikanth

M.Tech.,(Ph.D)

Asst.Prof, Dept of IT, VIT

## UNIT – V

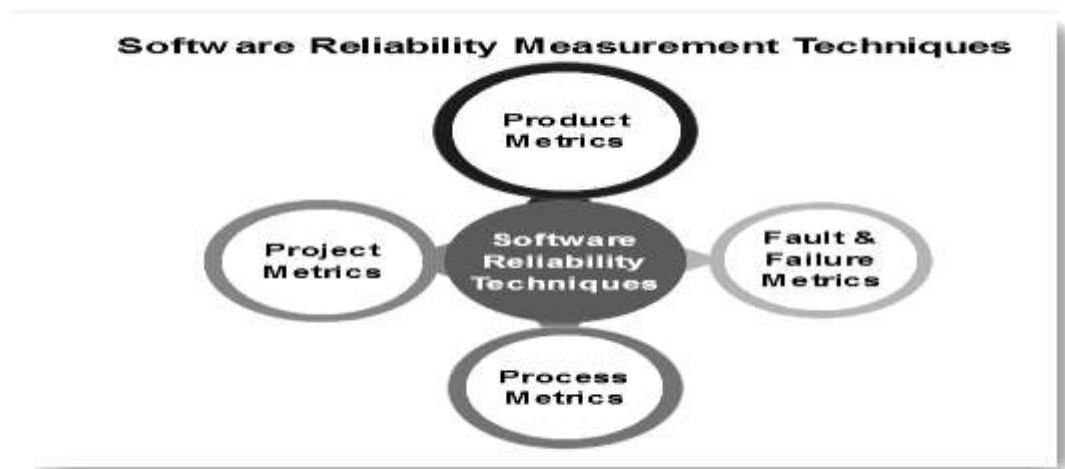
### SOFTWARE RELIABILITY AND QUALITY MANAGEMENT

#### SOFTWARE RELIABILITY

- Software reliability is defined as the probability of failure free operation of a complete program in a specified environment for a specified time.
- Categorizing and specifying the reliability of software systems.
- Probability of failure free operation for a specified time in a specified environment for a given purpose
- This means quite different things depending on the system and the users of that system
- Informally, reliability is a measure of how well system users think it provides the services they require
- What is the definition of software reliability? Technically speaking, the definition of software reliability is the probability of a failure-free operation over some period of time. However, it's been used to describe a collection of development practices aimed at improving software reliability or reducing software defects. There are software reliability models for predicting failure rate, defect density, availability during every part of the software lifecycle. Reliability definition within software engineering means simply that the software you develop performs its function as specified.

Two terms related to software reliability

1. **Fault:** A defect in the software, Ex. A bug in the code which may cause a failure
2. **Failure:** A derivation of the program observed behavior from the required behavior.



## STATISTICAL TESTING

- Testing software for reliability rather than fault detection
- Test data selection should follow the predicted usage profile for the software.
- Measuring the number of errors allows the reliability of the software to be predicted
- An acceptable level of reliability should be specified and the software tested and amended until that level of reliability is reached.

### Statistical testing procedure

- Determine operational profile of the software
- Generate a set of test data corresponding to this profile
- Apply tests, measuring amount of execution time between each failure
- After a statistically valid number of tests have been executed, reliability can be measured.

## SOFTWARE QUALITY

The quality of software can be defined as the ability of the software to function as per user requirement. When it comes to software products it must satisfy all the functionalities written down in the SRS document.

### Key aspects,

- **Good design** – It's always important to have a good and aesthetic design to please users
- **Reliability** – Be it any software it should be able to perform the functionality impeccably without issues
- **Durability**- Durability is a confusing term, in this context, durability means the ability of the software to work without any issue for a long period of time.
- **Consistency** – Software should be able to perform consistently over platform and devices
- **Maintainability** – Bugs associated with any software should be able to capture and fix quickly and new tasks and enhancement must be added without any trouble
- **Value for money** – customer and companies who make this app should feel that the money spent on this app has not gone to waste.



## SOFTWARE QUALITY MANAGEMENT SYSTEM

Quality software refers to software which is reasonably bug or defect free, is delivered in time and within the specified budget, meets the requirements and/or expectations, and is maintainable. In the software engineering context, software quality reflects both **functional quality** as well as **structural quality**.

- **Software Functional Quality** – It reflects how well it satisfies a given design, based on the functional requirements or specifications.
- **Software Structural Quality** – It deals with the handling of non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability, and the degree to which the software was produced correctly.
- **Software Quality Assurance** – Software Quality Assurance (SQA) is a set of activities to ensure the quality in software engineering processes that ultimately result in quality software products. The activities establish and evaluate the processes that produce products. It involves process-focused action.
- **Software Quality Control** – Software Quality Control (SQC) is a set of activities to ensure the quality in software products. These activities focus on determining the defects in the actual products produced. It involves product-focused action.





ISO (International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization. ISO declared its 9000 series of standards in 1987. It serves as a reference for the contract between independent parties. The ISO 9000 standard determines the guidelines for maintaining a quality system. The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc. ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself. The ISO 9000 series of standards is based on the assumption that if a proper stage is followed for production, then good quality products are bound to follow automatically. The types of industries to which the various ISO standards apply are as follows.

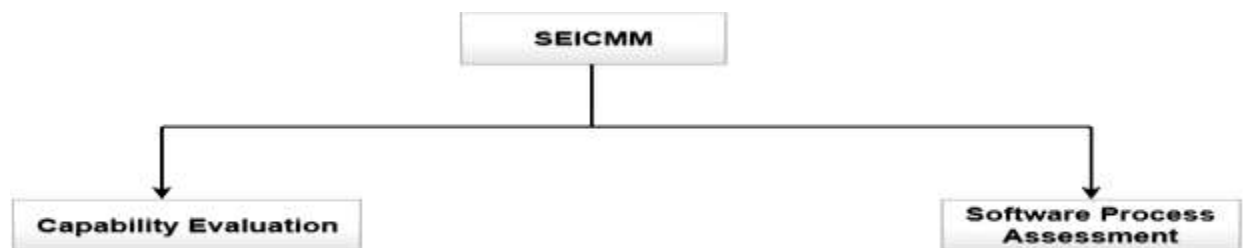
1. ISO 9001: This standard applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that applies to most software development organizations.
2. ISO 9002: This standard applies to those organizations which do not design products but are only involved in the production. Examples of these category industries contain steel and car manufacturing industries that buy the product and plants designs from external sources and are engaged in only manufacturing those products. Therefore, ISO 9002 does not apply to software development organizations.
3. ISO 9003: This standard applies to organizations that are involved only in the installation and testing of the products. For example, Gas companies.



## SEI- CAPABILITY MATURITY MODEL.

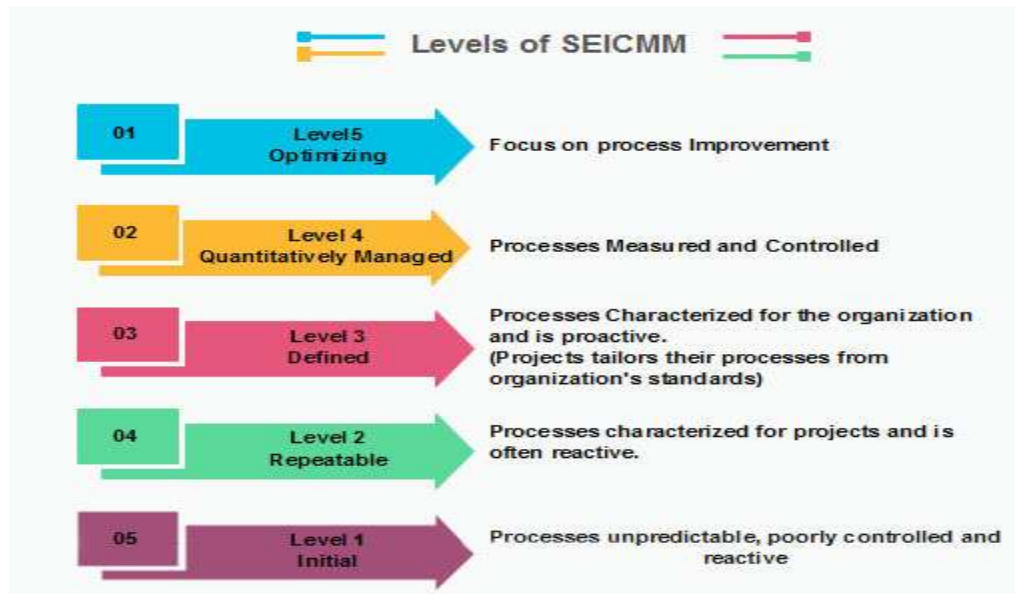
- The Capability Maturity Model (CMM) is a procedure used to develop and refine an organization's software development process.
- The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes.
- CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center promote by the U.S. Department of Defense (DOD).
- Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.

**There are two methods of SEICMM:**



**Capability Evaluation:** Capability evaluation provides a way to assess the software process capability of an organization. The results of capability evaluation indicate the likely contractor performance if the contractor is awarded a work. Therefore, the results of the software process capability assessment can be used to select a contractor.

**Software Process Assessment:** Software process assessment is used by an organization to improve its process capability. Thus, this type of evaluation is for purely internal use. SEI CMM categorized software development industries into the following five maturity levels. The various levels of SEI CMM have been designed so that it is easy for an organization to build its quality system starting from scratch slowly.



**Level 1: Initial:** Ad hoc activities characterize a software development organization at this level. Very few or no processes are described and followed. Since software production processes are not limited, different engineers follow their process and as a result, development efforts become chaotic. Therefore, it is also called a chaotic level.

**Level 2: Repeatable:** At this level, the fundamental project management practices like tracking cost and schedule are established. Size and cost estimation methods, like function point analysis, COCOMO, etc. are used.

**Level 3: Defined:** At this level, the methods for both management and development activities are defined and documented. There is a common organization-wide understanding of operations, roles, and responsibilities. The ways through defined, the process and product qualities are not measured. ISO 9000 goals at achieving this level.

**Level 4: Managed:** At this level, the focus is on software metrics. Two kinds of metrics are composed.

**Level 5: Optimizing:** At this phase, process and product metrics are collected. Process and product measurement data are evaluated for continuous process improvement.

## SOFTWARE MAINTENANCE

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updating done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

### Types of maintenance

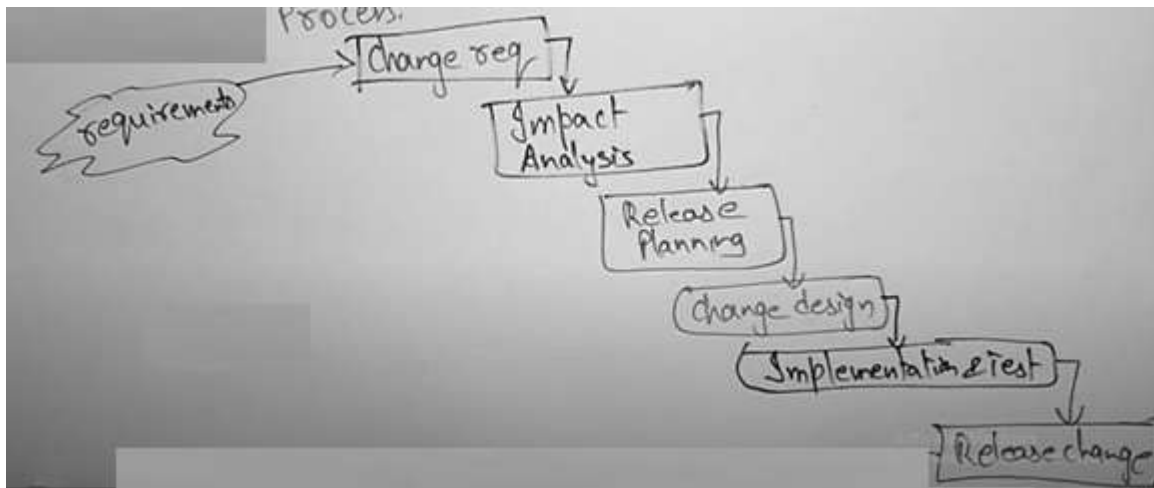
In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
- **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
- **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

M.Srikanth,  
M.Tech., (Ph.D)  
Asst.Prof, Dept of IT, VIT

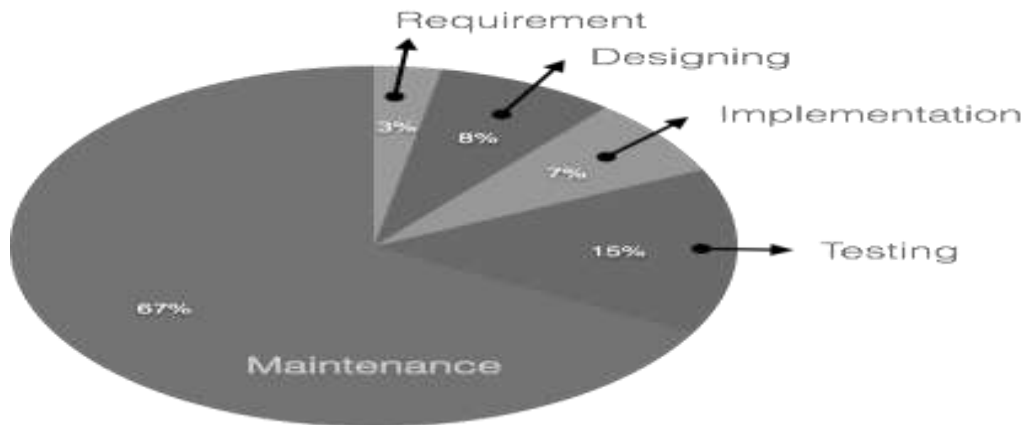
## MAINTENANCE PROCESS MODELS

- Maintenance process very considerably depending on the type of software being maintained
- The most expensive part of software life cycle is software maintenance process
- Two broad categories of process models for software maintenance can be proposed
- The first model is preferred for projects involving small reworks where the code is changed directly and the changes are reflected in the relevant document later. This maintenance process is graphically presented process model-1.
- The second process model for software maintenance is preferred for projects where the amount of rework required is significant. This maintenance process is graphically presented in process model-2.



## MAINTENANCE COST

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

### Real-world factors affecting Maintenance Cost

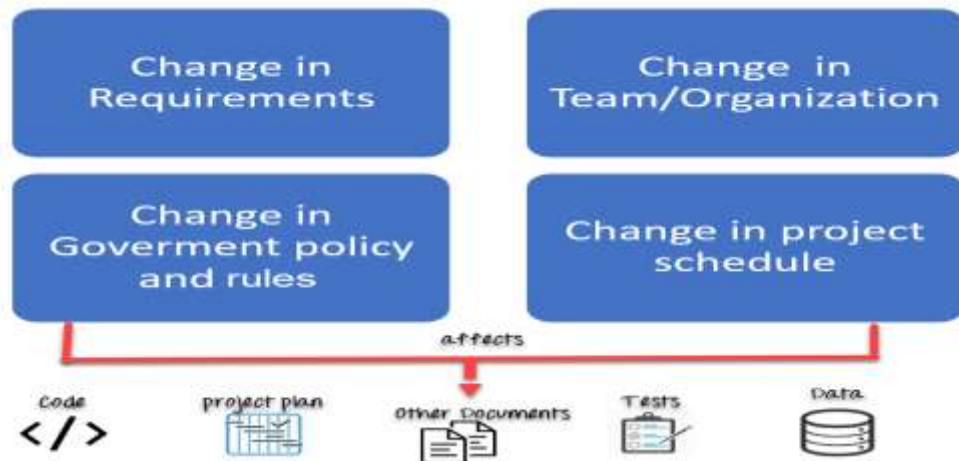
- The standard age of any software is considered up to 10 to 15 years.
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

### Software-end factors affecting Maintenance Cost

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

## SOFTWARE CONFIGURATION MANAGEMENT

In software engineering, software configuration management (SCM or S/W CM) is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. SCM practices include revision control and the establishment of baselines.



- Configuration Management best practices helps organizations to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.
- The primary goal of the SCM process is to increase productivity with minimal mistakes
- The main reason behind configuration management process is that there are multiple people working on software which is continually updating. SCM helps establish concurrency, synchronization, and version control.
- A baseline is a formally accepted version of a software configuration item
- Change control is a procedural method which ensures quality and consistency when changes are made in the configuration object.
- Configuration status accounting tracks each release during the SCM process
- Software Configuration audits verify that all the software product satisfies the baseline needs
- Project manager, Configuration manager, Developer, Auditor, and user are participants in SCM process
- The SCM process planning begins at the early phases of a project.