SWAMI DAYANANDA COLLEGE OF ARTS & SCIENCE MANJAKKUDI.

III-BCA/CS/IT

SUBJECT NAME :SOFTWARE ENGINEERING

SUBJECT CODE :16SMBECA1:2/16SMBECS1:1 / 16SMBEIT1:1

Dr. T. Nagarathinam

Assistant Professor, Department of CS

Subject Name : SOFTWARE ENGINEERING

Classes : III-BCA/ CS / IT

Subject Code : 16SMBECA1:2/16SMBECS1:1 / 16SMBEIT1:1

SOFTWARE ENGINEERING

Objectives:

- To provide knowledge of the various phases of Software Engineering Process
- To provide knowledge of different software development model and software testing methodologies.
- To learn about web engineering and emerging trends in software engineering.

Unit I

Introduction : Introduction to Software Engineering - Software Process - Software Process Models - Software Model - Requirements Engineering Principles : Requirements Engineering - Importance of Requirements - Types of Requirements - Steps involved in Requirements Engineering

Unit II

Requirements Analysis Modeling : Analysis Modeling Approaches - Structured Analysis - Object Oriented Analysis - Design and Architectural Engineering : Design Process and Concepts - Basic Issues in Software Design - Characteristics of Good Design - Software Design and Software Engineering - Function Oriented System vs Object Oriented System - Modularity, Cohesion, Coupling, Layering - Real Time Software Design - Design Models - Design Documentation

Unit III

Object Oriented Concepts : Fundamental Parts of Object Oriented Approach - Data Hiding and Class Hierarchy Creation - Relationships - Role of UML in OO Design - Design Patterns - Frameworks - Object Oriented Analysis - Object Oriented Design - User Interface Design : Concepts of User Interface - Elements of User Interface - Designing the User Interface - User Interface Evaluation - Golden Rules of User Interface Design - User Interface Models – Usability

Unit IV

Software Coding - Introduction to Software Measurement and Metrics - Software Configuration - Project Management Introduction - Introduction to Software Testing - Software Maintenance

Unit V

Web Engineering : Introduction to Web - General Web Characteristics - Web Application Categories - Working of Web Application - Advantages and Drawbacks of Web Applications - Web Engineering - Emerging Trends in Software Engineering - Web 2.0 - Rapid Delivery - Open Source Software Development - Security Engineering - Service Oriented Software Engineering - Web Service - Software as a Service - Service Oriented Architecture - Cloud Computing - Aspect Oriented Software Development - Test Driven Development - Social Computing

Textbook: 1. Software Engineering, Chandramouli Subramanian, SaikatDutt, Chandramouli Seetharaman, B.G.Geetha, Pearson Publications, 2015.

Reference Books: 1. Software Engineering,Jibitesh Mishra, Pearson Education, 2011.

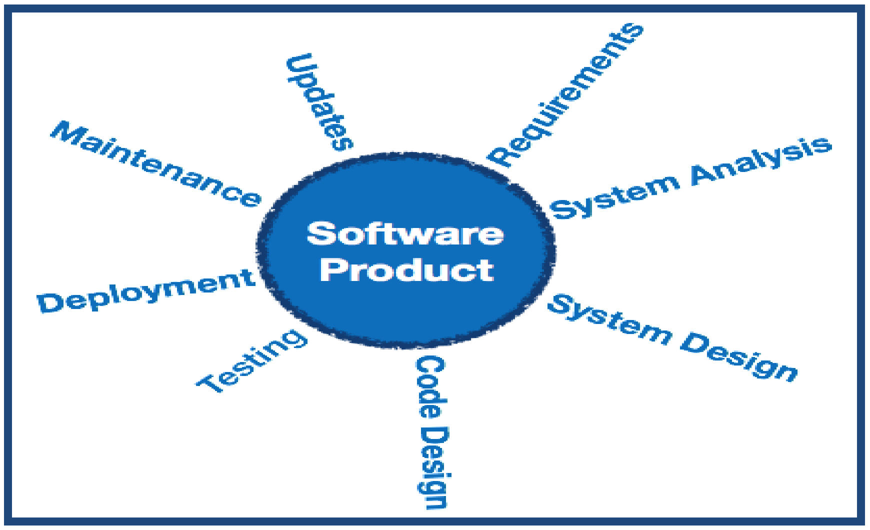
SOFTWARE ENGINEERING

Unit - 1

- ❖ Introduction to software engineering
- ❖ Software process
- ❖ Software process models
- ❖ Software model
- ❖ Requirement engineering principles: requirement engineering
- ❖ Importance of requirements
- ❖ Types of requirements: Steps involved in requirements engineering

Software:
Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**
Engineering:
Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software Engineering:
Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.



IEEE Definition:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.

INTRODUCTION TO SOFTWARE ENGINEERING:

- The success of mankind in performing a process with perfection and accuracy commenced with the origin of computers.
- The computer performed simple tasks and produced results.
- The primary components of the computer included a memory space to store the variables, a procedure to perform the operations and a display to show the results to the user.
- A programming language has certain formats to guide the user for easier and faster designing of the program.

1.What is software?

- Software is defined as the tool with designed order of instruction, performing tasks to provide the desired results after obtaining the requirements of the users.
- Software also consists of procedures, rules, software-related documents and the associated basic data required to run the software.
- **Software=computer program+ procedures+ rules+ documents +data**

Examples of software:

- 1.MS-Office
- 2.Linux
- 3.MYSQL
- 4.Adobe Photoshop

2.Evolving Role of Software:

- Instruction-> Software has evolved from a single instruction provided to the machine to the present stage of decision-making software.
- Programs-> programs perform an operation, collectively they form software and perform a function for the user.
- Software-> software is classified into two types based on the areas they command, namely, application software and system software.
- System software-> control the operations of the hardware components.
- Examples: operating system (DOS, Windows and Linux) Device drivers.
- Application software-> it consists of programs purely for the tasks of the tasks of the users, the input and processing them in the instructions already defined.
- Examples: MS-Office, MS calculator, Notepad, etc…
- Verifier-> other types of software which are dedicated to the verification of the programs in terms of coding, flow of control and order of execution.
- Examples: compilers and debuggers.

3.Phases in software development:

- The phases in software development define the steps of organizing the development of the software.
- Each step is called a phase.

The general software development phases:

1.Requirements Analysis phase
2.Designing phase
3.Coding and Testing phase
4.Implementation phase
5.Maintenance phase

1.Requirement Analysis

- The requirement analysis is the first phase of software development wherein the customer and the business analysts interact with each other and document the requirement of the outcome of the software.

Types of requirement :

- Functional requirements
- Non-functional requirements
- Interface requirements

The steps in Requirement Engineering phase:

- Feasibility study
- Eliciting requirements
- Requirements analysis
- Specify and validate requirement
- Requirement management

2.Designing

- The software requirement is converted into design models.
- The design of the software product should:
- 1.Be precise and specific
- 2.Conform to budget and requirements
- 3.Provide a direct solution to the problem
- Software design includes data design, architectural design, user interface design, procedural design and deployment level design.

3.Coding and Testing

- Design is transformed into lines of codes in programming language.
- After coding phase is completed, the code have to tested for ensuring its right functionality in the platform.

4.Implementatin:

- This phase involves the release of the developed software into the market and to the end users. This phase is also called as deployment phase.
- The success of the implemented product depends on how much the end users like the product and prefer to use it regularly.

5.Maintenance:

- Software maintenance is the core aspect of software engineering. The process of modifying the production system after the delivery to correct the faults, improve the performance and adapt to the changing environment is called as software maintenance.

Changing Nature of software:

- Binary instruction-> It were fed into the computer machine.
- Development tools-> the software has evolved to a matured level and has sophisticated tools to develop the software.
- Object-based development-> Now a days object-based for software development in use.
- Open source software-> it is a big hit in the market and anybody can change the code while using and customizing it as per the need.

- Drag and Drop customizations-> the users to design a software product themselves by dragging and dropping the components at their intended positions.

1.In-built software

- The in-built software are deeply sensitive and efficient, prescribing the intelligence functions of the services. The decision-making skill sets. The reducing the participation of the mankind in in-human conditions.

2.System-based software

- The system-based software has also some prescribed functions, but they are limited to simpler devices. The system-based software is related to programs, which define the functionality of the hardware components of a particular computer.

3.Application – based software

- They obtain an input condition or a query from the end user, analyze it and display the results on the monitor.

6.Software Myths:

- The software development brackets many myths. The myths have created an incomplete and false image of the software development processes.

1.Software is better than implementing devices

- Devices have been limited to the wear and tear prone or usage. Software does not come with an expiry date, but practically there is a limit to its usage. The changes are easier to be updated in the software than devices.

2.Reusability ensures better solution

- The conceptual model of reusing a component is an incredible strategy to improve the performance reduce the time in designing and implementing the software.

3.Additional Features add efficiency

- The product being developed needs to have a direct and dedicated solution to the present problem.

4.More designers- more delay

- This is absolutely not true because the development team comprises of engineers with great knowledge on the strategies of rapid development.

8.Why study software engineering?

- Software engineering encompasses every concepts and standards of the disciplines in the relation to design, coding and developing maintainable software complying with the available resources and budget constraints.

An assessment of software engineering terminologies:

- It is the basic of any software development
- It helps to produce reliable, reusable and quality software
- Software engineering understands the customer needs and develop the software
- It helps the developer to understands the disciplines of software life cycle
- It helps to develop software in efficient way
- Software engineering teaches the best practices of software development
- Software engineering helps to write software, which can be integrated easily with other software.

9.Generic view of software engineering:

- The sequential steps that describe the stage of a product being developed are the generic view of software engineering.

Three generic phases:

1.Definition phase-> The desire outcomes, functionalities and objectives are obtained from the customers and documented.

2.Description phase-> The desire outcomes are framed with optimal preceding and succeeding processes. The framework of data and control flow in a product, how the processes are ordered and how the functionalities are defined are described in this phase.

3.Development phase-> They have to be transformed into the corresponding codes and implemented in the real-world environments. The training provided to the customers and end users, maintenance and upgrade at regular intervals.

10.Role of Management in software engineering

- Management of the software products depends on the following factors:

  1.Participants
  2.Procedure
  3.Product

1.Participants->

- Customers-> the customers are the initial participants of a product.
- Team members-> obeying the technical and administrative constraints.
- Team Leads-> It responsible for every process of software development.
- Manager-> It is also responsible for managing scope, time, cost and quality of the overall project

2.Procedures-> Procedure describes the way in which the product is being developed. The order and types of procedures are devised by the team members.

- The final plan to be implemented is approved by the team manager.
- The life cycle that has to be followed among the various models.\
- 3.Product-> The selection of best plan and procedures, and assigning the right and efficient staff to the project.
  Ensuring the final outcome of the project called as product.

SOFTWARE PROCESS

- A set of interrelated actions and activities to achieve a predefined result is called as process.
- Process has its own set of inputs and produces the outputs. Groups of individual processes constitute whole software.
- Overview of different software development processes:

  1.The Linear sequential model
  2.A layered technology
  3.Prototyping model
  4.RAD model
  5.Process Framework
  6.Capability maturity model integration
  7.Process Pattern
  8.Process Assessments

1.The Linear sequential model

- The sequence of actions describes the order in which the execution of activities are planned and followed.

- Linear sequential model was the initial step in developing a software product, otherwise known to be the lifecycle model.
- The process of development is divided into sequence of actions from requirements analysis, designing, coding and testing, implementation and finally maintenance.
- Linear sequential model is also called predictive life cycle model and classical life cycle model.

Advantages of linear sequential model:

- Easy to understand
- Easy for implementation
- Identifies deliverables and milestones upfront

Disadvantages of linear sequential model:

- Requirements are elaborated over a period of time.
- Takes longer time of finish the projects

Which is difficult in long-term project

2.A Layered Technology

- Similar to the linear sequential model, a layered approach ensures a much stronger product.
- The layered model of software engineering divides the activities into four categories:
- 1.Quality process-> There should be a strong base of quality processes.
- 2.Process-> Ensure that all the technology layers work together and enable the timely development of the software system.
- 3.Methods-> It provide the technical capabilities to the system. The requirements analysis, design, modeling, development, testing etc… are tasks related to the methods.
- 4.Tools-> It provide the support to the process and methods by making the tasks automated.
- For example: the roads are layered with four or five layers.
- The bottom layer is a combination of clay and large stones, followed by another layer of smaller stones and tar.
- The quality of the roads depends on the tar used, the number of times run over by a roller machine and the time given for the settlement of those combinations.

3.Prototyping Model:

- The product once developed using layered approach cannot be reversed easily.
- The prototyping model was proposed with a completely different strategy.
- Every product to be developed initially receives the requirement from the customers.
- A sample or test model called the prototype is developed and displayed to the customer.
- The prototype is altered until the customer is satisfied on the style, design and execution of the processes.

4.RAD Model

- Rapid Application Development (RAD) model is the methodology proposed for quicker design and deployment of a product.
- RAD model requires a very short time span of 60-90 days for completing the software and delivering it to the customer.
- RAD model proves to be the most preferable solution to quick needs and implementation of web-based application.

Five phases for development:

1.Business modeling
2.Data modeling
3.Process modeling
4.Application modeling
5.Testing and turnover

1.Business Modeling

Business modeling is the stage in which the types of information that are prime factors of the application are defined.

2.Data modeling

The obtained information from the previous phase is filtered into useful and meaningful sets of data.

3.Process modeling

Process modeling defines the activities needed to process the entities. The initial to the final stages of the application.

4.Application Generation

Application generation is a phase that uses automated tools to generate the working model of the designed application.

5.Testing and Turnover

The final phase is to test the correctness and consistency of the developed application.

Process Framework:

- Framework activities are processes of basic functionalities and are common to almost all software products.
- For example, a user login form always commences further session of every user. The login form consists of only two requirements: **username and password.**
- Analysis on the process framework activities:
- **Gathering the requirements**-> The customary process framework activities begin with the analysis and observations made on the requirements. This process called as communication phase involves intended customers and the team of development.
- **Scheduling and staff allocation**-> Based on the gathered information and the plans the model is selected for development, processes are framed and the analysis on the feasible risks are made.
- **Design and Deployment**-> The team members start designing and developing the coding for their assigned task. The design process may be eliminated by reusing a module.
- **Supplementary Framework Activities**-> The product being developed has to be checked at regular intervals to ensure that it is proceeding in the right path.

Capability maturity model Integration

- Capability Maturity Model Integration (CMMI) is an approach for improving the process level operations.
- The CMMI is a compilation of many best approaches to enhance a products efficiency, quality and endurance.

CMMI heeds about three areas:

1.Product and Service Development
2.Service Establishment and Management
3.Product and Service Acquisition

CMMI Levels:

- The CMMI model was evolved from the software CMM for integrating many different and well-organized models into new models.

There are five levels of model that are evolutionary:

1.Initial
2.Repeatable
3.Defined
4.Managed
5.Optimizing

- Initial Level-> it is a poorly controlled phase with no proper designation of the schedule and the resources needed for the development.
- Repeatable -> Yet those processes are in urge of additional consistency in the repeatable level.
- Defined Level-> It requires stringent measures to be followed. The processes are regulated, well understood and properly ordered.
- Manage Level-> It is the fourth level in which adequate steps are taken to determine the actions to update and upgrade the processes in the product.
- Optimizing Level-> This level forwards its outcome to the optimizing level which concentrates on smoothening of approaches for better, faster and risk-free execution of the processes.

Process Pattern:

- There has been a standard order of activities for completing a task.
- Patterns compile a *set of activities or tasks or actions*, following a prescribed sequence of execution.
- These are patterns are considered in software lifecycles, customer communication, coding, reviews and test patterns.

Process Assessments:

- Every process has its **own strength, weakness** and associated risks.
- Evaluation of the strengths, weaknesses and risks of every process against a process model is called as Process Assessment.
- Process Assessment describes the results of evaluation as:
- Incomplete-> Still needs revision and modifications
- Performed-> Complete definition and satisfies the customer
- Managed-> Controlled and maintained
- Established-> Standard procedure applied
- Predictable-> Defined with limits of execution
- Optimizing-> Slight alternations required

Software Product

1.Characteristics of A Good Software Product

- Completeness -> It should cover all stated and agreed requirements without missing out any.
- Consistency -> It should be stable and capable of handling the problems in implementation
- Durability-> Customers may vary in technology requirement.

- Efficiency -> The product should not waste the resource, execution and waiting time of operations and the memory spaces allocated for the processes.
- Security-> It should be confidential and high significance.
- Interoperability-> Communication between other processes.

<u>Engineering Aspects of Software Product</u>

- A software product has different views of understanding. For technical and engineering aspects of software product, the customers see an abstract image of the product.

  <u>1.Purpose:</u>
  - The aim of a software product needs a strong justification. The necessary conditions for the product to work.

  <u>2.Process:</u>
  - Definition of the operations needs engineering skills for the right placement and interrelationships.

<u>3.Role of Management in Software Products</u>

- Management plays an important role in driving the processes of development in the right path benefiting the customer and the organization.

  <u>1.Defining the Vision</u>
  - The vision is the force that motivates the development team. The vision keeps the team on track to maintain the standards under any condition.

  <u>2.Defining the Plan</u>
  - The plans are the operation derived with inputs and desires on the outputs. The plan encompasses many individual processes, structured in an ordered sequence.

  <u>3.Defining the Design</u>
  - Every participating member in a development process is assigned with a role and jobs to perform.

  <u>4.Testing the Pathway</u>
  - Testing is an important process in managing the defined plans. Testing includes the verification of the correctness and consistency of the product, whether budget and time constraints are met.

  <u>5.Supportive Activities</u>
  - Licensing the product as the registered outcome of a particular organization and marketing the product among the competitive organizations are some of the additional and adequate activities, which support the delivery of the software product.

<u>**Software Process Models**</u>

Software process models are systematic methods for controlling and coordinating the development of a software product achieving all the stated objectives.

<u>Software process models help the developer team to recognize the following:</u>

1.Set of tasks, subtasks and interfaces
2.Resources needed for every process
3.Adequate time span

Software Process model types

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

1. **Waterfall Model**

   Waterfall model is the first and foremost methodology of Software Development Life Cycle (SDLC). The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks. The approach is typical for certain areas of engineering design .

The original methodology consisted of the following activities:

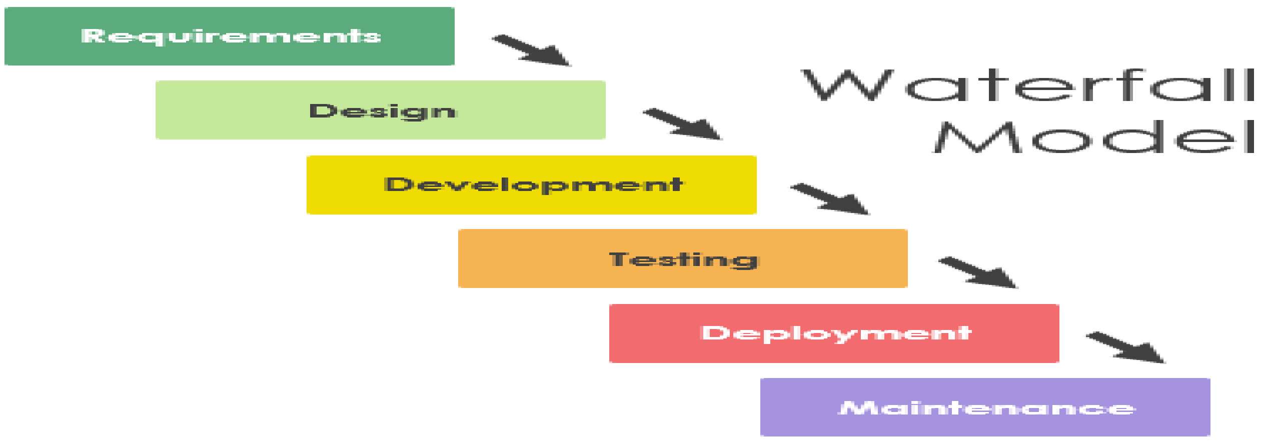
- Requirements specification(phase1), Design (phase2), Construction(phase3), Integration (phase4), Testing (phase5), Debugging(phase6), Installation(phase7) and Maintenance(phase8).
- Analysis is obviously the first process that leads the development team.
- The development team will start designing a framework for achieving the objectives.
- The software needs additional features after deployment based on the demand from users and customers.
- All the processes of the waterfall model ensure in sequential order.

Advantages of Waterfall model:

- Easy to understand
- Easy to implementation
- Widely used and known
- Identifies deliverables and milestones upfront

Disadvantages of waterfall model:

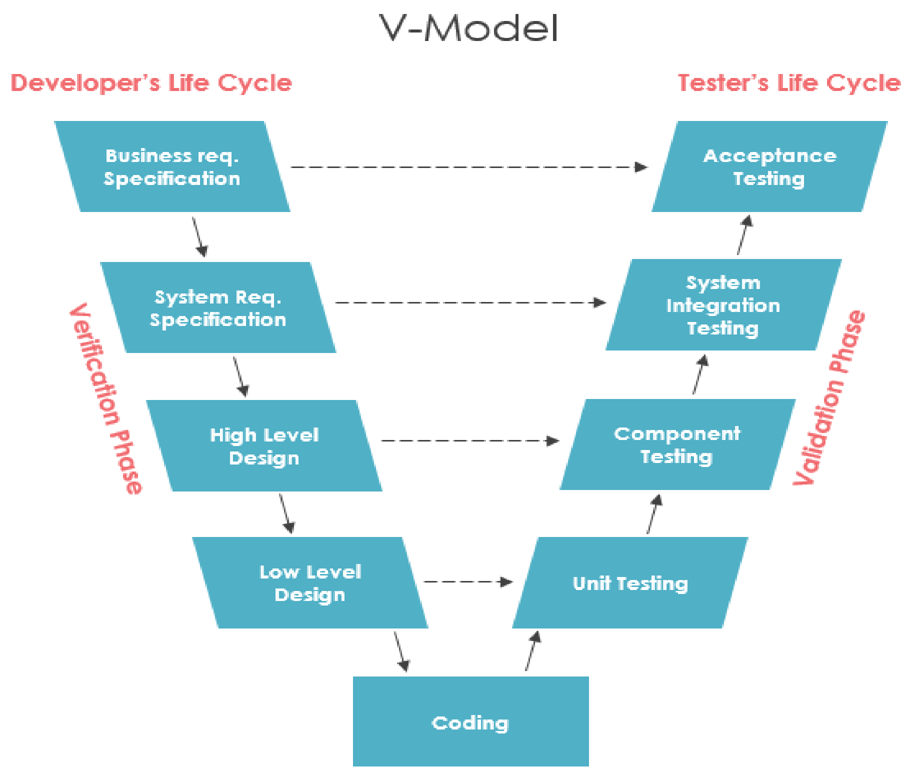
- Does not match well with reality
- It is unrealistic to freeze the requirement upfront
- Software is delivered only at the last phase
- Difficult and expensive to make changes



V Model

The V-model represents a development process that may be considered an extension of the waterfall model and is an example of the more general V-model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape.

The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

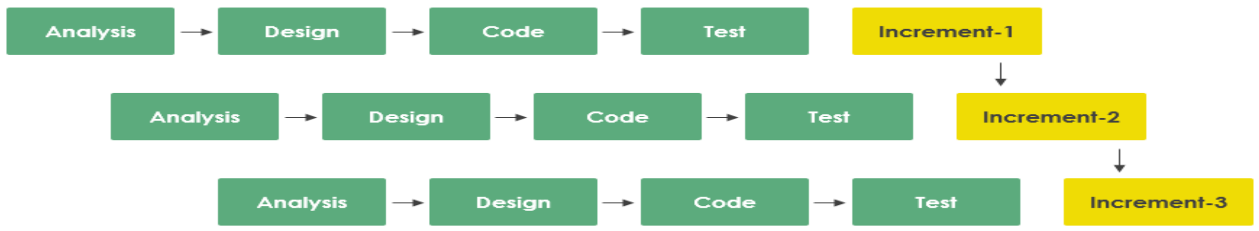


Incremental model

The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished. It involves both development and maintenance.

The product is defined as finished when it satisfies all of its requirements. Each iteration passes through the requirements, design, coding and testing phases.

And each subsequent release of the system adds function to the previous release until all designed functionally has been implemented. This model combines the elements of the waterfall model with the iterative philosophy of prototyping



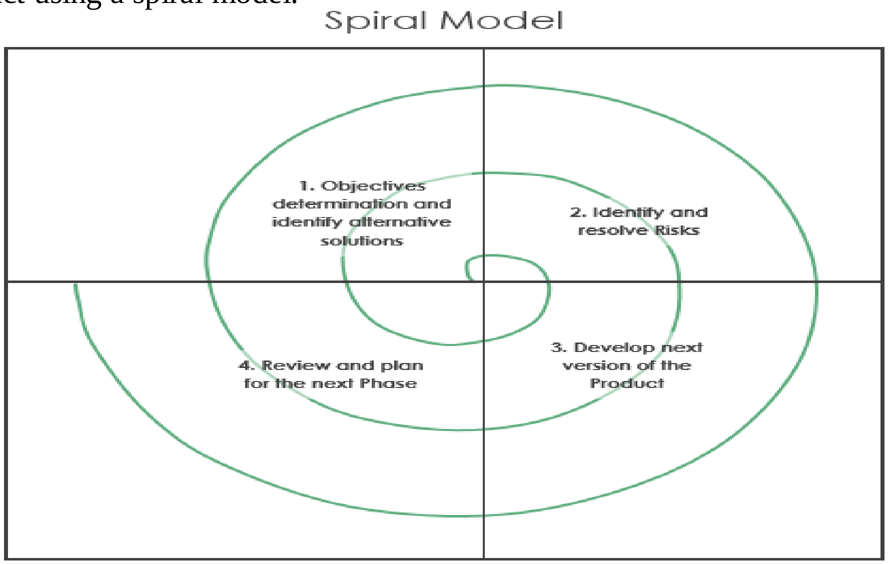
Incremental Model

Spiral model

The spiral model, first described by Barry Boehm in 1986, is a risk-driven software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model. A spiral model looks like a spiral with many loops.

The exact number of loops of the spiral is unknown and can vary from project to project. This model supports risk handling, and the project is delivered in loops. Each loop of the spiral is called a Phase of the software development process.

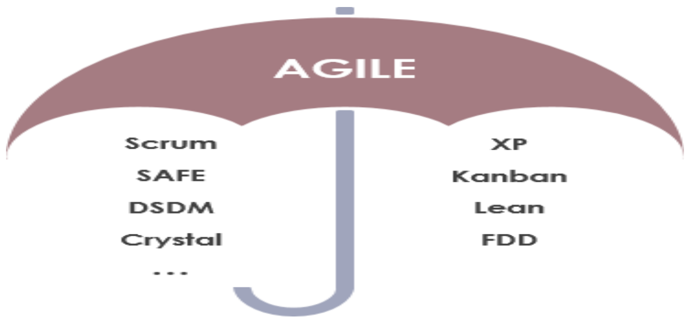
The initial phase of the spiral model in the early stages of Waterfall Life Cycle that is needed to develop a software product. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using a spiral model.


Spiral Model
1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

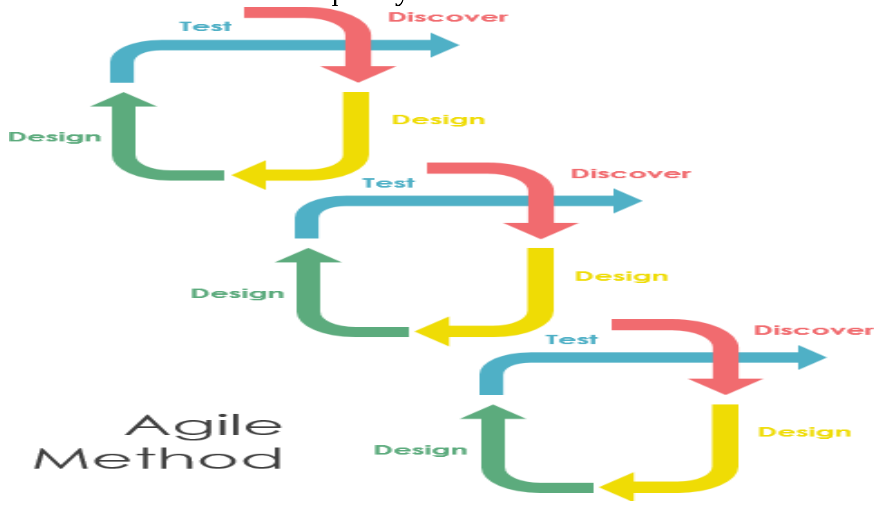
Agile model

Agile is an umbrella term for a set of methods and practices based on the values and principles expressed in the Agile Manifesto(is a document that identifies 4 key values and 12 principles that its authors believe software development should use to guide their work.) that is a way of thinking that enables teams and businesses to innovate, quickly respond to changing demand, while mitigating risk.

Organizations can be agile using many of the available frameworks such as Scrum, Kanban, Lean, Extreme Programming (XP) and etc.



The primary goal of being Agile is empowered the development team the ability to create and respond to change in order to succeed in an uncertain environment.
Agile software development approach is typically operated in rapid and small cycles. This results in more frequent incremental releases with each release building on previous functionality. Thorough testing is done to ensure that software quality is maintained.



Principle 1: Our Highest Priority is to **Satisfy the Customer** through **Early** and **Continuous Delivery** of Valuable Software
Principle 2: **Welcome Changing Requirements**, even late in Development. Agile Processes harness change for the Customer's Competitive Advantage.
Principle 3: Deliver Working Software Frequently
Principle 4: **Build Projects around Motivated Individuals**. (Give them the environment and support they need, and trust them to get the job done.)
Principle 5: The Most Efficient and Effective Method of Conveying Information to and within a Development Team is **face-to-face Communications.**
Principle 6: Working Software is the Primary Measure of Progress
Principle 7: **Agile Processes promote sustainable develop** The sponsors, developers, and users should be able to **maintain a constant speed** indefinitely.
Principle 8: Continuous Attention to Technical Excellence and Good Design enhances Agility.

REQUIREMENTS ENGINEERING PRINCIPLES

- The software project life cycle starts with capturing the requirements of the project.

What is Requirements Engineering?

- Requirement is defined as "a condition or capability needed by a user to solve a problem or processed by a system.
- Requirement engineering is the discipline that explores the externally imposed conditions on a proposed computer system and tries to identify the capabilities that will meet those imposed conditions and recording the same in the form of documentation called the requirement document of the computer system.

Importance of Requirements

- Requirements are the stepping stones to the success of any project.
- In software development, primary focus is usually given to the construction phase, which leads a lot of problems at the end.
- A software project has to be completed within a specified time frame and budget.

- 3 times the cost to fix it during the design stage
- 5 to 10 times the cost to fix it during the construction stage
- 10 times the cost to fix it during the system testing stage of the project and
- 10 to 100 times the cost to fix it during the post release phase.

<u>Types of Requirements</u>

- It can be categorized into three main types:

  1.Functional requirements,

  2.Nonfunctional requirements and

  3.Interface requirements

<u>1.Functional requirements</u>

- The set of requirements that defines what the system will do or accomplish is called functional requirements.
- The requirements may be for performing calculations, data manipulation, processing, logical decision-making .

<u>2.Non-functional Requirements</u>

- The quality attributes and the design and architecture constraints that the system must have are called nonfunctional requirements.
- Some examples of NFR:
- The system should be available 24X7 (availability)
- The system should save or fetch 1000 records in 1 second (performance)
- Product-related NFR include performance, availability, maintainability, portability, reliability, security, scalability, testability and usability.
- Measuring NFR: we need to first define measurable criteria for each NFR and then realize the metrics by measuring it.
- Approaches to NFR: NFR can be approached in two ways- product-oriented and process –oriented approaches.

<u>3.Interface Specification</u>

- Need to interact with other systems in order to work. They interact with many other systems to receive and send data, get help in processing logic, store information in other systems.

<u>Steps involved in requirements engineering</u>

- Requirements engineering includes requirements development and requirement management.

The basic aims of the requirement engineering process are to provide a mechanism to understand the user's needs, analyze the need, feasibility, specify and validate the requirements and proposed solution, and manage the requirements over the whole life cycle of the project.

- <u>The following section:</u>
  - Feasibility study
  - Requirement elicitation
  - Requirement analysis
  - Specifying and validating requirements
  - Requirements management

<u>1.Feasibility study</u>

- Feasibility study is the first step of the requirements engineering process.
- <u>Feasibility can be classified into three categories:</u>

- Operational feasibility-> Checks the usability of the proposed software. If the operational scope is high, then the proposed system will be used more ensuring the acceptance from the sponsors of the project.
- Technical feasibility-> Checks whether the level of technology required for the development of the system is available with the software firm, including hardware resources, software development platforms and other software tools.
- Economic feasibility-> Checks whether there is scope for enough return by investing in this software system. All the costs involved in developing the system, including software licenses, hardware procurement and manpower cost, needs to be considered while doing this analysis.

Requirement Elicitation

- Requirements elicitation is the second step of the requirements engineering process.
- The source for all the requirements are identified and the user's needs and all the possible problem statements are identified.

The stakeholders involved during this phase include end users, managers, development and test engineers, domain expert and business analysts

1.Identifying Stakeholder

- Before the requirements are gathered for a system, it is important to know the people who can contribute and help gather the requirements.
- Stakeholders are people or entities actively involved in the projects.

2.Characteristics of Stakeholder

- Stakeholder interests may be either positively or negatively impacted by the performance of the project. Stakeholders may have an influence on the project and its results.

3.Problems in Eliciting requirements

- There are several problems which can surface while eliciting requirements,
- Users not sure about the requirements
- Communication gap
- Conflicting requirements
- Volatile requirements

Requirements Elicitation Techniques

- Several techniques can be employed for eliciting the requirements as listed below.
    - Interviewing
    - Focus groups
    - Facilitated workshops
    - Prototyping
    - Questionnaires
    - Brainstorming(idea collection)
    - Direct observation
    - Apprenticing

Requirements Analysis

- This is the phase that bridges the gap between requirements engineering and design.

The focus areas for the analyst during this stage are:

- Externally observable data objects
- Flow and content of information
- Software functionality elaboration
- Behavior of the software on an external input

    - Interface requirements
    - Design constraints

Specifying Requirements

- These steps in a clear, concise and unambiguous manner so that the design and development teams can use it going forward. This is called the requirement specification phase.

1.Specifying requirement in the SRS Document

- The SRS document can be used for several purposes:
- Agreement between the customer and the suppliers.
- The design and development team start their work based on this document.
- The resourcing, budgeting, scheduling and pricing for the project are based on the volume.
- A good SRS ensures future maintainability of the project.

2.IEEE standards and SRS Template

- The aim of the SRS document is to provide an understanding of what the stakeholders want from the system and how the system should behave to cater to these needs.

3.Checklist for writing a Good SRS Document

- The consumers of the SRS document are all the stakeholders of the project—the customer, end users, project engineers, interface developers , managers.
- Checkpoints lists:
    - Correctness
    - Completeness
    - Consistency (constant)
    - Verifiability
    - Clarity
    - Priority
    - Modifiability

5.Validating Requirements

- The main aim of requirements validation is to ensure that the customer needs are captured completely, clearly and consistently.

The different stakeholder who should be involved in the validation process:

- Customer
- End user
- Domain expert
- Architect
- Construction and test engineer
- Third-party system interacting with the proposed system

There are different approaches taken for validating the requirements:

- Walkthrough
- Review and
- Inspection
- Models
- Prototypes

Requirement Management

- How the existing requirement can be stored and tracked and how the changes to these requirements.

Requirement management plan

The plan for managing the requirements should mention how to manage the following:

- Requirements identification
- Requirements change
- Requirements status tracking
- Requirements traceability

- **Requirement identification**-> this creates the baseline set of requirements on which the implementation team starts working.
- **Requirements change**-> Requirements are bound to change during the course of time.
- **Requirements status tracking**-> The set of requirement get base lined, which can be modified after the change control board approves a requirement for implementation.
- **Requirements Traceability**-> The most commonly used artifact to track the requirements in each stage of the project life cycle such as design, construction, unit testing and system testing is a traceability matrix that maintains the tracing information.

Unit II

Requirement Analysis modeling

Requirements Analysis Modeling :

- ❖ Analysis Modeling Approaches
- ❖ Structured Analysis
- ❖ Object Oriented Analysis

Design and Architectural Engineering :

- ❖ Design Process and Concepts
- ❖ Basic Issues in Software Design
- ❖ Characteristics of Good Design
- ❖ Software Design and Software Engineering

1.Analysis Modeling Approaches

- The analysis stage acts as the bridge between design and requirement.
  The following objectives:
  - Clearly
  - Explain the functional activities
  - Classify the design problem
  - Define system behavior and class structure
  - Define data flow

$$\begin{tikzcd} & \text{Analysis modeling approaches} \arrow[dl] \arrow[dr] & \\ \bullet\ \text{Structured analysis} & & \text{Object oriented analysis} \end{tikzcd}$$

1.1 Data modeling

1.2 Functional or flow oriented modeling

1.3 Behavior modeling

1.Structured Analysis

The domain of structured analysis three modeling approaches,

1.1.Data modeling
1.2.Functional modeling
1.3.Behavioral modeling

1.1.Data modeling

- Data modeling is an analysis technique that deals with the data processing part of an application.
- Deals with identifying the data elements in a system, the structure and composition of data, relationship and different processes that causes transformation to these data.

1.1.1Data objects, Attributes and Relationship:

- Data object or an entity in a system is the identity which has multiple attributes and is related to other entities in the system.

1.1.2 Cardinality and Modality

- How many types of each object can be attached with other objects, which is called cardinality.

- Whether it is mandatory to attach an object with the another object type which is called modality.

1.1.3.Entity Relationship Diagrams

- The relationships of the objects present in a system are represented graphically through the ERD.
- The entities are represented as a rectangle.
- The relationships are represented with lines with the ends representing the cardinality and modality.

The various components of an ERD:

- Data object or entities
- Attributes
- Relationships and various indicators
- 1.4.Data Dictionary
- Data dictionary is the place where the description and information about all data objects produced and consumed by the software system is maintained.

1.2.Funcitonal modeling

- Functional modeling, data flow diagram(DFD)
- Functionality flow is also know as the data flow.

1.2.1 Data Flow Diagram

- Entire system is shown as a process and data are interlinked to each other.
- The four component:
    - Entities
    - Processes
    - Data stores
    - Data flows
- 2.2.Control flow model
- There are systems where the flow is controlled by events rather than by data.

1.3. Behavioral modeling

The type of analysis models which try to capture the change in behavior of the system as an effect of some event or trigger are grouped as the behavioral models.

3.1.1.State Transition Diagram(STD)

- STD represents the system states and events that trigger state transitions or state change.

3. Object-Oriented Analysis

The attributes of the objects are represented by its class, the state and the behavior

Design and Architectural Engineering

1.Design Process and concepts

- The software requirement model, which was prepared for the requirements is converted into suitable and appropriate design models that describe the architecture and other design components.

2.Basic issues in software design

- If the requirement model is wrongly represented then the design model also will go wrong.
- Mapping each functional requirement into design representation is difficult.
- Balancing simplicity and efficiency is difficult.
- Balancing clarity and code safety is difficult.

- Error function handling is also a difficult of design.

3.Characteristics of a good design

- Should be able to convert all the requirement into design
- Should be easily understandable and maintainable
- Should be easily to change the design
- Should be easily scalable

4.Software design and software engineering

- Software design should also be assessed for its own quality parameters such as robustness, flexibility, security, and design clarity.
- Software engineering comprises strict disciplines that need to be followed to develop a programmable solution to solve the problems of customers.

5.Function-oriented system Vs Object-oriented system

- The design of function-oriented system is called function-oriented design and the design of object-oriented system is called object-oriented design.
- Data are stored outside the system in a function-oriented system.
- Functions are used to access, process, modify the data and store it back into the data storage system.
- Object-oriented design is the design concept of an object-oriented system, which is completely made up of objects.
- Objects have data and functions inside them.

6.Modularity, Cohesion, Coupling, Layering

- Modularity: The processing of dividing a single project into smaller units called modules is termed modularity of the project.
- Cohesion: It refers to how strongly one module is related to the other, which helps to group similar items together.
- Types of cohesion:
    - Coincidental
    - Logical
    - Temporal
    - Procedural
    - Communication
    - Sequential
    - Functional
- Coupling: It is a measure of “how tightly“ two entities or modules are related to each other.
- Types of coupling:
    - Content
    - Common
    - Control
    - Stamp
    - Data

7.Real-time software design

- Real time refers to designing the software systems whose behavior is subject to timing constraints and is embedded in a large hardware system.
- This kinds of software monitors and controls the system and its environment by gathering different sets of data using sensors.

1.Design Process and concepts

- The software requirement model, which was prepared for the requirements, is converted into suitable and appropriate design models that describe the architecture and other design components.

Introduction to design process

The main aim of design engineering is to generate a model. Software design is an iterative process through which requirements are translated into the blueprint for building the software.

Software quality guidelines

- A design is generated using the **recognizable architectural styles** and compose a
- **good design characteristic of components** and it is implemented in evolutionary manner for testing.
- A design of the software must be modular i.e the software must be logically partitioned into elements.
- In design, the representation of data , architecture, interface and components should be distinct.
- A design must carry appropriate data structure and recognizable data patterns.
- Design components must show the independent functional characteristic.
- A design creates an interface that reduce the complexity of connections between the components.
- A design must be derived using the repeatable method.
- The notations should be use in design which can effectively communicates its meaning.

Quality attributes

The attributes of design name as 'FURPS' are as follows:

Functionality:

It evaluates the feature set and capabilities of the program.

Usability:

It is accessed by considering the factors such as human factor, overall aesthetics, consistency and documentation.

Reliability:

It is evaluated by measuring parameters like frequency and security of failure, output result accuracy, the mean-time-to-failure(MTTF), recovery from failure.

Performance:

It is measured by considering processing speed, response time, resource consumption, throughput and efficiency.

Supportability:

It combines the ability to extend the program, adaptability, serviceability.

These three term defines the maintainability.

Testability, compatibility and configurability are the terms using which a system can be easily installed and found the problem easily.

Supportability also consists of more attributes such as compatibility, extensibility, fault tolerance, modularity, reusability, robustness, security, portability, scalability.

Design concepts

The set of fundamental software design concepts are as follows:

1. Abstraction

A solution is stated in large terms using the language of the problem environment at the **highest level abstraction**.

The **lower level of abstraction** provides a more detail description of the solution.

A sequence of instruction that contain a specific and limited function refers in a **procedural abstraction.**

A collection of data that describes a data object is a **data abstraction.**

2. Architecture

The complete structure of the software is known as software architecture.

Structure provides conceptual integrity for a system in a number of ways.

The architecture is the structure of program modules where they interact with each other in a specialized way.

The components use the structure of data.

The aim of the software design is to obtain an architectural framework of a system.

The more detailed design activities are conducted from the framework.

3. Patterns

A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Modularity

Software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.

Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding. The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion

Cohesion is an extension of the information hiding concept.

A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

Coupling is an indication of interconnection between modules in a structure of software.

7. Refinement

Refinement is a top-down design approach.

It is a process of elaboration.

A program is established for refining levels of procedural details.

A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statements are reached.

8. Refactoring

It is a reorganization technique which simplifies the design of components without changing its function behavior.

Refactoring is the process of changing the software system in a way that it does not change the external behavior of the code still improves its internal structure.

9. Design classes

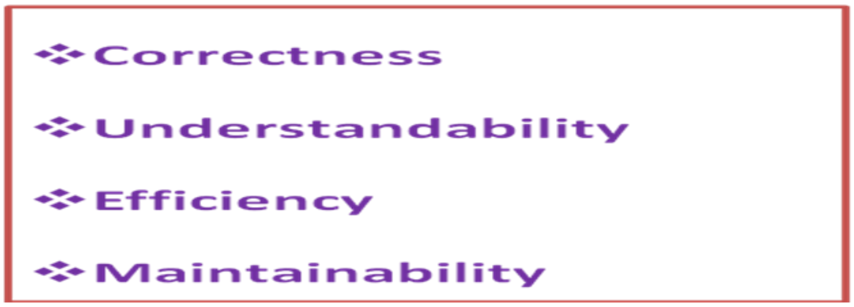
The model of software is defined as a set of design classes.

Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

<u>Characteristics of a good design</u>

- Should be able to convert all the requirement into design
- Should be easily understandable and maintainable
- Should be easily to change the design
- Should be easily scalable

For good quality software to be produced, the software design must also be of good quality



Correctness

Understandability

Efficiency

Maintainability

1) **Correctness**

First of all, the design of any software is evaluated for its correctness.

The evaluators check the software for every kind of input and action and observe the results that the software will produce according to the proposed design.

If the results are correct for every input, the design is accepted and is considered that the software produced according to this design will function correctly.

2) Understandability

The software design should be understandable so that the developers do not find any difficulty to understand it.

Good software design should be self- explanatory. This is because there are hundreds and thousands of developers that develop different modules of the software, and it would be very time consuming to explain each design to each developer.

So, if the design is easy and self- explanatory, it would be easy for the developers to implement it and build the same software that is represented in the design

3) Efficiency(Capacity to do things right which is given in the requirement)

The software design must be efficient. The efficiency of the software can be estimated from the <u>design phase</u> itself, because if the design is describing software that is not efficient and useful, then the developed software would also stand on the same level of efficiency.

Hence, for efficient and good quality software to be developed, care must be taken in the designing phase itself.

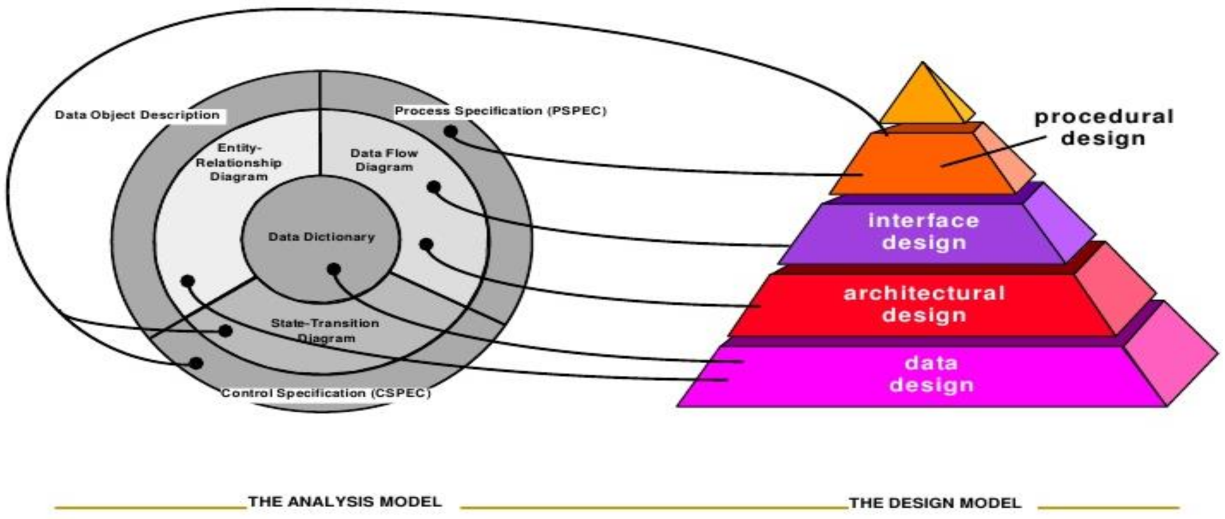
4) Maintainability

The software design must be in such a way that modifications can be easily made in it. This is because every software needs time to time modifications and maintenance.

So, the design of the software must also be able to bear such changes. It should not be the case that after making some modifications the other features of the software start misbehaving. Any change made in the software design must not affect the other available features, and if the features are getting affected, then they must be handled properly.

Software design and software engineering

- Software design should also be assessed for its own quality parameters such as robustness, flexibility, security, and design clarity.
- Software engineering comprises strict disciplines that need to be followed to develop a programmable solution to solve the problems of customers.



Abstraction

A solution is stated in large terms using the language of the problem environment at the highest level abstraction.

The lower level of abstraction provides a more detail description of the solution.

A sequence of instruction that contain a specific and limited function

A collection of data that describes a data object is a data abstraction

Architecture

The complete structure of the software is known as software architecture.

Structure provides conceptual integrity for a system in a number of ways.

The architecture is the structure of program modules where they interact with each other in a specialized way.

The components use the structure of data.

The aim of the software design is to obtain an architectural framework of a system.

The more detailed design activities are conducted from the framework.

Design Patterns

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Uses of Design Patterns

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

Often, people only understand how to apply certain software design techniques to certain problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem.

In addition, patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

Information hiding

Information hiding is an important aspect of modularity, and if you recall the definition of abstraction (reducing information content to only what is important), information hiding is an important aspect to the abstraction of software.

Specifically, consider that the final software system is the lowest level of abstraction. All of the software's design details are present at this level. Information hiding allows us to hide information unnecessary to a particular level of abstraction within the final software system, allowing for software engineers to better understand, develop and maintain the software.

We use software modules to implement information hiding: the information contained in the modules should be hidden from those the rest of the software system outside of the module, and access to this hidden information should be carefully controlled. This allows us to maintain a higher level of abstraction in our software, making our software more comprehensible.

If information hiding is done well, changes made to the hidden portions of a module should not affect anything outside of the module. This allows the software engineers to more readily manage change (including changes in the requirements).

Function-oriented system Vs Object-oriented system

The design of function-oriented system is called function-oriented design and the design of object-oriented system is called object-oriented design.
Data are stored outside the system in a function-oriented system.
Functions are used to access, process, modify the data and store it back into the data storage system.
Object-oriented design is the design concept of an object-oriented system, which is completely made up of objects.
Objects have data and functions inside them.

Function-oriented design

The following are the salient features of a typical function-oriented design approach:

1. A system is viewed as something that performs a set of functions. Starting at this high-level view of the system, each function is successively refined into more detailed functions.

For example, consider a function create-new library-member which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge.

This function may consist of the following sub functions:

• Assign-membership-number
• Create-member-record
• Print-bill

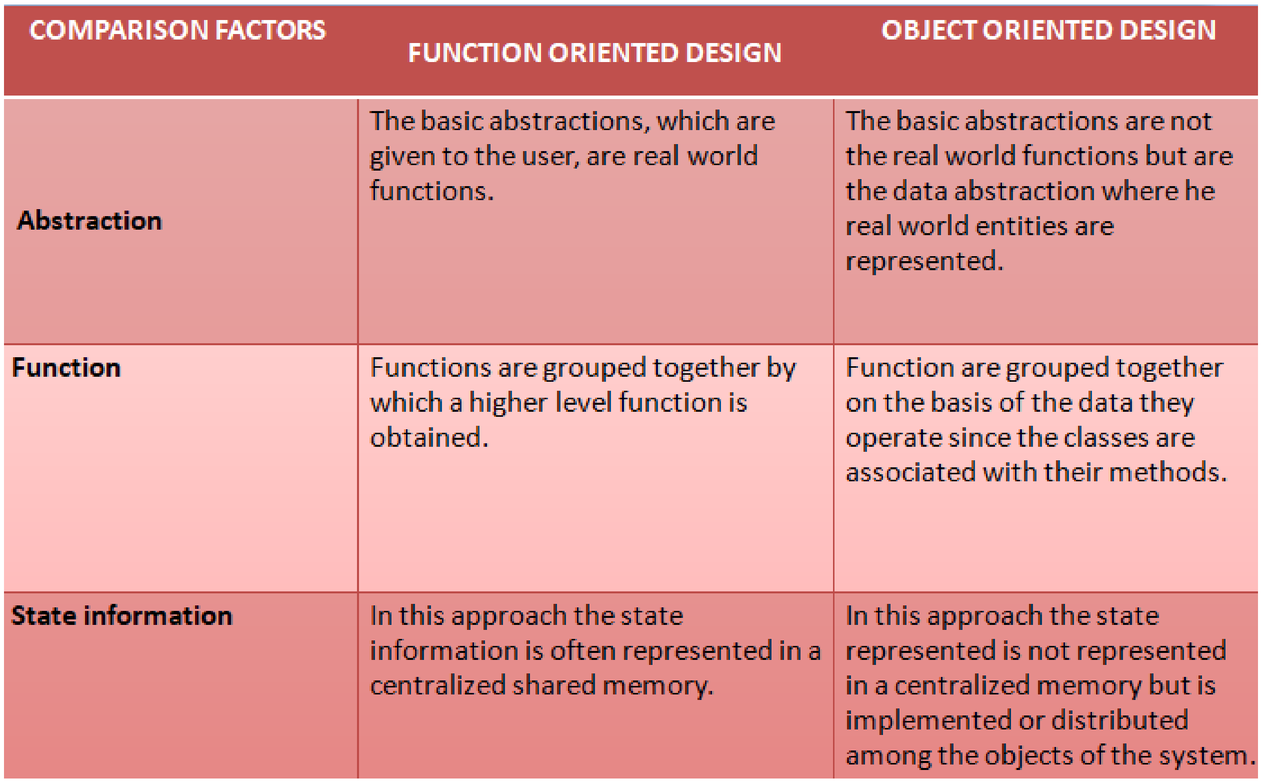
Each of these sub-functions may be split into more detailed sub functions and so on.

2. The system state is centralized and shared among different functions, e.g. data such as member-records is available for reference and updating to several functions such as:

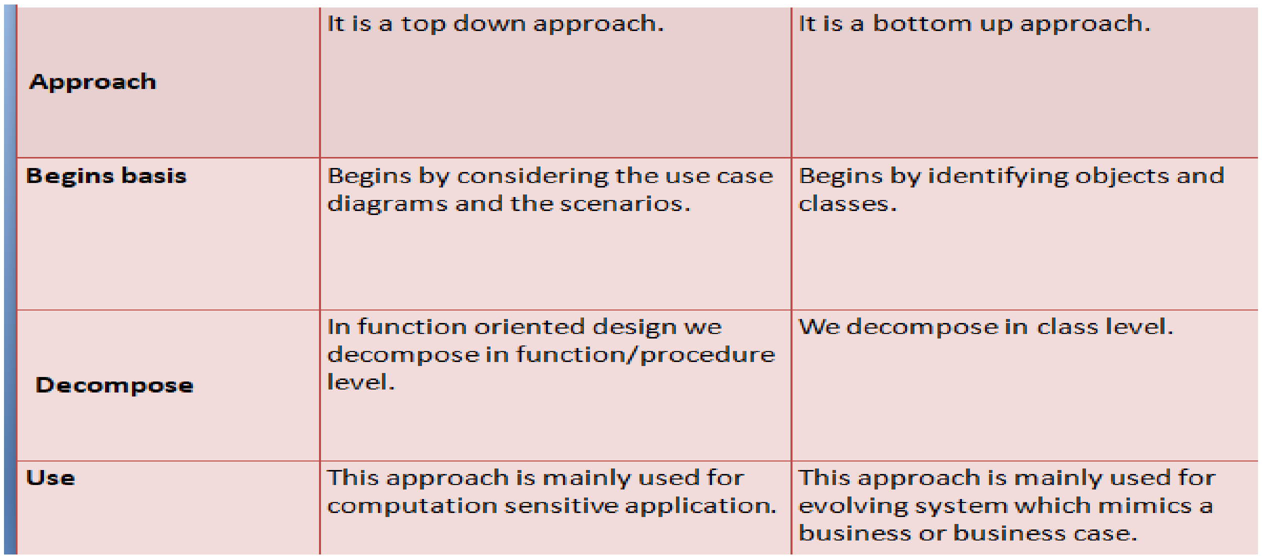
- create-new-member
- delete-member
- update-member-record

Object-oriented design

In the object-oriented design approach, the system is viewed as collection of objects (i.e. entities).
The state is decentralized among the objects and each object manages its own state information. For example, in a Library Automation Software, each library member may be a separate object with its own data and functions to operate on these data.
In fact, the functions defined for one object cannot refer or change data of other objects. Objects have their own internal data which define their state. Similar objects constitute a class. In other words, each object is a member of some class. Classes may inherit features from super class. Conceptually, objects communicate by message passing.



COMPARISON FACTORS	FUNCTION ORIENTED DESIGN	OBJECT ORIENTED DESIGN
Abstraction	The basic abstractions, which are given to the user, are real world functions.	The basic abstractions are not the real world functions but are the data abstraction where he real world entities are represented.
Function	Functions are grouped together by which a higher level function is obtained.	Function are grouped together on the basis of the data they operate since the classes are associated with their methods.
State information	In this approach the state information is often represented in a centralized shared memory.	In this approach the state represented is not represented in a centralized memory but is implemented or distributed among the objects of the system.



Approach	It is a top down approach.	It is a bottom up approach.
Begins basis	Begins by considering the use case diagrams and the scenarios.	Begins by identifying objects and classes.
Decompose	In function oriented design we decompose in function/procedure level.	We decompose in class level.
Use	This approach is mainly used for computation sensitive application.	This approach is mainly used for evolving system which mimics a business or business case.

Modularity, Cohesion, Coupling, Layering

Modularity can be defined as a mechanism where a complex system is divided into several components that are referred to as 'modules'. Now, whenever a customer requests to develop software we can use or integrate these modules to develop new software.

The required modules can be selected and integrated to develop new software in this way, we can customize the software according to users need.

Advantages of modularization

1. Smaller components are easier to maintain.
2. Program can be divided based on functional aspects.
3. Desired level of abstraction can be brought in the program,
4. Re-use of components
5. Concurrent execution can be made possible.
6. Ease to change
7. Ease to build
8. Ease to maintain.

Cohesion

A good software design implies clean decomposition of the problem into modules and the neat arrangement of these modules in a hierarchy. The primary characteristics of neat module decomposition are **low coupling** and **high cohesion.**
Cohesion is a measure of functional strength of a module. A module having **low coupling and high cohesion** is said to be **functionally independent** of other modules.
Functional independence means that a cohesive module performs a single function or task. A functionally independent module has very little interaction with other modules.
Types of Cohesion

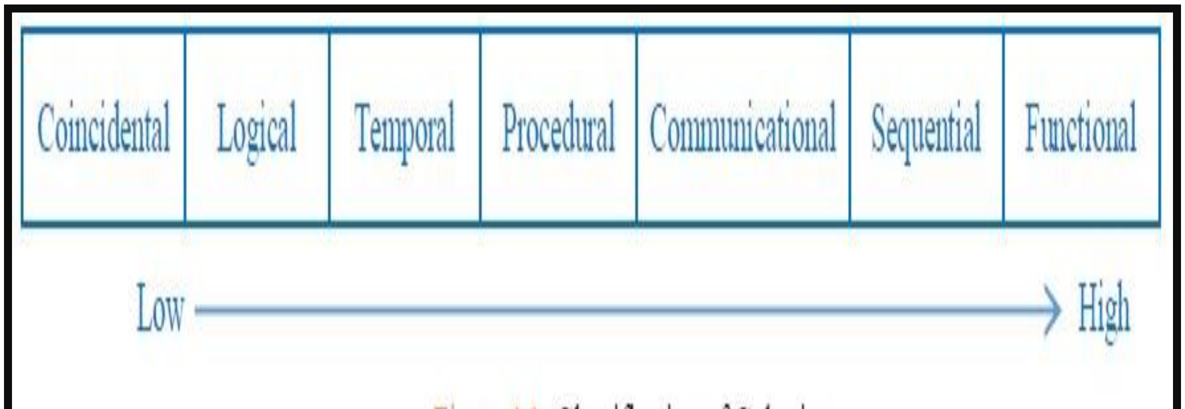


Figure 4.1: Classification of Cohesion

Coincidental Cohesion:
A module is said to have coincidental cohesion if it performs a set of function or tasks that relate to each other very loosely.
In this case, the module contains a random collection of functions. It is likely that the functions have been put in the module out of pure coincidence without any design or thought.
For example, in a transaction processing system (TPS), the get-input, print-error, and summarize members functions are grouped into one module. The grouping does not have any relevance to the structure of the problem.
Logical Cohesion:
A module is said to be logically cohesive, if all elements of the module perform similar operations, e.g. error handling, data input, data output, etc.
An example of logical cohesion is the case where a set of print functions generating different output reports are arranged into a single module.
Temporal Cohesion:
When a module contains functions that are related by the fact that all the functions must be executed in the same time span, the module is said to exhibit temporal cohesion.
The set of functions responsible for initialization, start-up, a shutdown of some process, etc. exhibit temporal cohesion.
Procedural Cohesion:
A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective, e.g. the algorithm for decoding a message.
Communicational Cohesion:
A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, e.g. the set of functions defined on an array or a stack.
Sequential Cohesion:
A module is said to possess sequential cohesion if the elements of a module form the parts of the sequence, where the output from one element of the sequence is input to the next.
For example, in a TPS, the get-input, validate-input, sort-input functions are grouped into one module.
Functional Cohesion:
Functional cohesion is said to exist if different elements of a module cooperate to achieve a single function.
For example, a module containing all the functions required to manage employees' pay-roll exhibits functional cohesion.
Suppose a module shows functional cohesion and we are asked to tell what the module does, then we would be able to tell it using a single sentence.

Coupling

Coupling between two modules is a measure of the degree of interaction or interdependence between the two modules. A module having low coupling and high cohesion is said to be **functionally independent** of other modules.
If two modules interchange huge amounts of data/information, then they are **highly interdependent**. The degree of coupling between two modules depends on their interface complexity, which is basically determined by the number of types of parameters that are interchanged while invoking the functions of the module.

Types of Coupling

The classification of the different types of coupling helps to quantitatively estimate the degree of coupling between two modules. Five types of coupling can occur between any two modules.

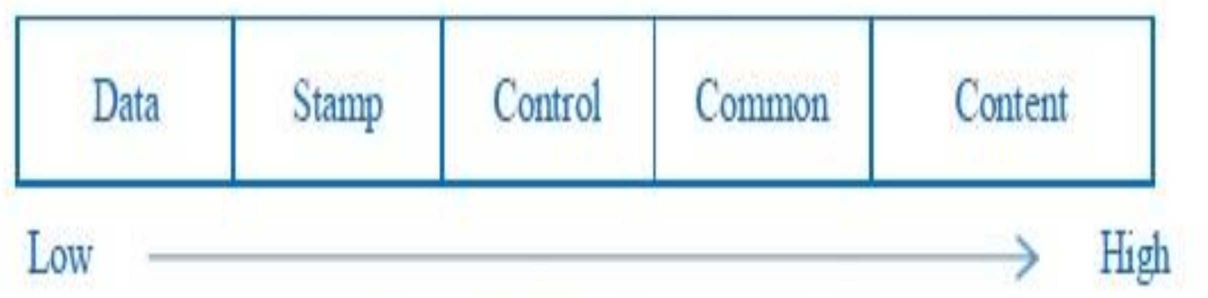


Figure 4.2: Classification of Coupling

Data Coupling:

Two modules are data coupled if they communicate through a parameter. An example is an elementary data item passed as a parameter between two modules, e.g. an integer, a float, a character, etc. This data item should be problem-related and not used for the control purpose.

Stamp Coupling:

Two modules are stamp coupled if they communicate using a composite data item such as a record in PASCAL or a structure in C.

Control Coupling:

Control coupling exists between two modules if data from one module is used to direct the order of instructions executed in another. An example of control coupling is a flag set in one module and tested in another module.

Common Coupling:

Two modules are common coupled if they share data through some global data items.

Content Coupling:

Content coupling exists between two modules if they share code, e.g. a branch from one module into another module.

<u>Real-time software(RTS) design</u>

- Real time refers to designing the software systems whose behavior is subject to timing constraints and is embedded in a large hardware system.
- This kinds of software monitors and controls the system and its environment by gathering different sets of data using sensors.
- Actuators change the environment of the system.

Microwave oven has a functionality to stop the cooking after a specified time period. In this case, the sensor continuously monitors the time and the actuator stops the control of the microwave oven after a specified time period.

8.Design models

1.Data Design

- In data design, a high-level model is depicted based on the user's view of the data or information.
- Data design is actually derived from the ER diagram.

2.Architectural Design

- Architectural design is derived from requirement analysis and is also based on already available architectural patterns and styles.

3.User Interface Design

- The user interface design creates an effective communication medium between a human and a computer.
- Three basic areas:

  1.interface between components of the system.
  2.interface between the system and humans
  3.interface between the system and other systems.
- The three "golden rules" of interfaces design are:

  1.place the user in control
  2.reduce the user memory control
  3.makes the interface consistent.

4.Procedural Design

- Procedural design is also called component design.
- Component design is usually done after user interface design.

5.Deployment-level design

- Deployment –level design takes care of the procedure of implementing the actual systems and their components within a physical environment.

9.Design Documentation

There are two types of design documents, namely, high-level design document and low-level design document.

- A high-level design document contains architectural details.
- System Architecture Document(SAD) is the design document where all the design and architectural element are captured.
- A low-level design document contains SAD.
- **High-level design** provides a view of the system at an abstract level. It shows how the major pieces of the finished application will fit together and interact with each other.
- A high-level design should also specify assumptions about the environment in which the finished application will run. For example, it should describe the hardware and software you will use to develop the application, and the hardware that will eventually run the program.
- The high-level design does not focus on the details of how the pieces of the application will work. Those details can be worked out later during low-level design and implementation.
- Before you start learning about specific items that should be part of the high-level design, you should understand the purpose of a high-level design and how it can help you build an application.

The Software Architecture Document

- It provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system.
- **Software Architecture Document Guidelines**

  An outline description of the **software architecture**, including major **software** components and their interactions.

  A common understanding of the **architectural** principles used during design and implementation.

  A description of the hardware and **software** platforms on which the system is built and deployed.

UNIT-III

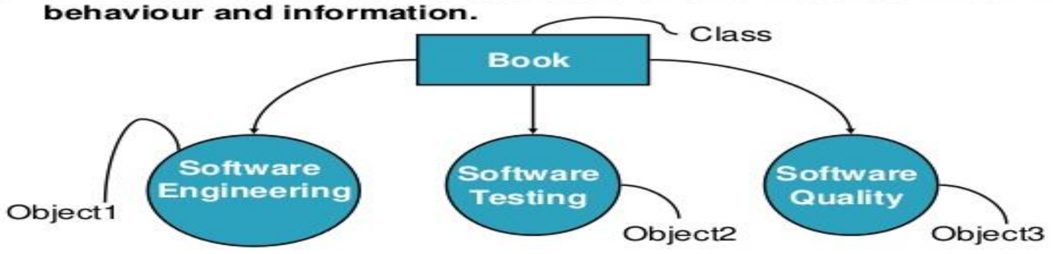
- Object Oriented Concepts :
- Fundamental Parts of Object Oriented Approach
- Data Hiding and Class Hierarchy Creation
- Relationships
- Role of UML in OO Design
- Design Patterns
- Frameworks
- Object Oriented Analysis
- Object Oriented Design
- User Interface Design :
- Concepts of User Interface
- Elements of User Interface
- Designing the User Interface
- User Interface Evaluation
- Golden Rules of User Interface Design
- User Interface Models
- Usability
- **Object-oriented concepts**
    - The object-oriented approach has made a rapid progress since the 1980.
    - If we think about the very basic building block of an O.O model.
    - **Object-oriented concepts** are used in the **design** methods such as classes, **objects**, polymorphism, encapsulation, inheritance, dynamic binding, information hiding, interface, constructor, destructor.
    - The main advantage of **object oriented design** is that improving the **software** development and maintainability.
- **Fundamental Parts of Object Oriented Approach**
    - In the object-oriented approach, the focus is on capturing the structure and behavior of information systems into small modules that combines both data and process.
    - The main aim of Object Oriented Design (OOD) is to improve the quality and productivity of **system analysis** and **design** by making it more usable.

The OO model is beneficial in the following ways −

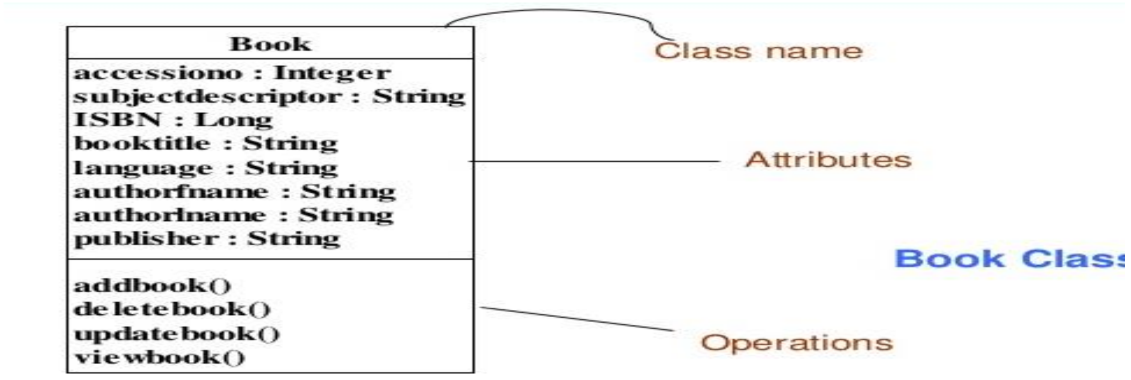
- It facilitates changes in the system at low cost.
- It promotes the reuse of components.
- It simplifies the problem of integrating components to configure large system.
- It simplifies the design of distributed systems.

Class: A class is the building block that leads to Object-Oriented Programming. It is a user-defined data type, that holds its own data members and member functions, which can be accessed and used by creating an instance of that class. It is the blueprint of any object.

- All book types may be combined to form a group called class.
- All objects are instances of a class.
- The class describes the structure of the instance which include behaviour and information.



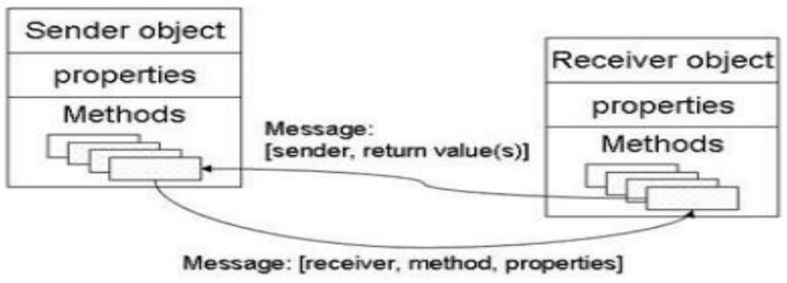
3.Attributes:
Properties or characteristics that represent the state of an object are called attributes.



4. Methods:
The method implements the behavior of an object. The collection of method that exhibits what the object is going to function is called the behavior.

5. Messages:
The objects interact with each other through messages.

Message Exchange in Classes



Relationships

Relationships between objects can be denoted in three forms namely,

1.is-a relationship
2.has-a relationship
3.uses-a relationship

1.is-a relationship: Inheritance is used to create an “is-a” relationship among classes. Generalization is also called as the “is-a” relationship.

2.has-a relationship: Aggregation is also called as the “has-a” relationship.

3.uses-a relationship: the method of one class may use the object of another class. This is called a uses-a relationship.

Role of UML in OO Design:

- UML stands for Unified Modeling Language.
- Used to models software and non-software system.
- Objects are real-world entities that exist around us.
- Object oriented basic concepts (object, class, inheritance, polymorphism, encapsulation etc.,) can be represented easily using UML.
- The OO design is converted into UML diagrams using UML notations.

Design Patterns

- A **design pattern** is a general repeatable solution to a commonly occurring problem in software design.
- A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Design patter has 4 essential parts:

- Pattern name
- Problem
- Solution
- Consequences

Uses of Design Patterns

- Design patterns can speed up the development process by providing tested, proven development paradigms.
- Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.
- Often, people only understand how to apply certain software design techniques to certain problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem.
- Patterns allow developers to communicate using well-known, well understood names for software interactions.

1. **Creational patterns**

   Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or added complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation.

- **Abstract Factory**
  Creates an instance of several families of classes
- **Builder**
  Separates object construction from its representation
- **Factory Method**
  Creates an instance of several derived classes
- **Object Pool**
  Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- **Prototype**
  A fully initialized instance to be copied or cloned
- **Singleton**
  A class of which only a single instance can exist

2. Structural patterns

- In Software Engineering, Structural Design Patterns are Design Patterns that ease the design by identifying a simple way to realize relationships between entities.
  - **Adapter**
    Match interfaces of different classes
  - **Bridge**
    Separates an object's interface from its implementation
  - **Composite**
    A tree structure of simple and composite objects
  - **Decorator**
    Add responsibilities to objects dynamically
  - **Facade**
    A single class that represents an entire subsystem
  - **Proxy**
    An object representing another object.

3.Behavioral design patterns

- These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

- **Chain of responsibility**
  A way of passing a request between a chain of objects
- **Command**
  Encapsulate a command request as an object
- **Interpreter**
  A way to include language elements in a program
- **Iterator**
  Sequentially access the elements of a collection
- **Mediator**
  Defines simplified communication between classes
- **Memento**
  Capture and restore an object's internal state
- **Observer**
  A way of notifying change to a number of classes
- **State**
  Alter an object's behavior when its state changes
- **Strategy**
  Encapsulates an algorithm inside a class
- **Template method**
  Defer the exact steps of an algorithm to a subclass
- **Visitor**
  Defines a new operation to a class without change

FRAMEWORK

- A framework is a platform for developing software applications.
- It is more specific solution description for design problems.
- A framework may consist of multiple design patterns.
- A framework may depicts the implementation of a whole system (it is a combinations of creation patter, structural and behavior pattern to form the whole system design and presented as a framework).

Difference between the Design pattern and Framework:

1. Design pattern more abstract level than framework.
2. Frameworks will be applicable only for specific system design but the design patterns are more generic and will be applicable to all situations for particular design problem.
3. Design patters are smaller design element than framework.
4. Design pattern are more generalized than framework.

Data hiding and class hierarchy creation

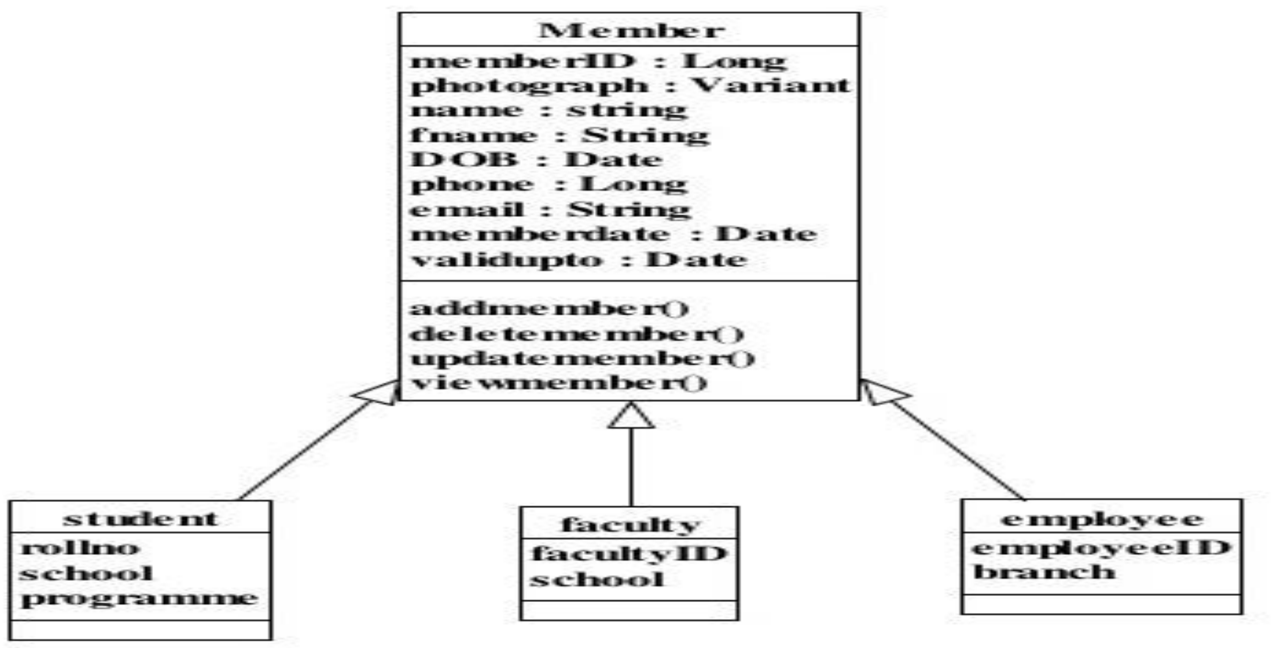
The concept of data hiding and class hierarchy creation leads to three major characteristics of the OO concept that differentiates it from the traditional design concepts:

- ❖ Encapsulation
- ❖ Inheritance
- ❖ Polymorphism

Encapsulation is defined as wrapping up of data and information under a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulates them.

Inheritance

The capability of a class to derive properties and characteristics from another class is called **Inheritance**. Inheritance is one of the most important feature of Object Oriented Programming.
Sub Class: The class that inherits properties from another class is called Sub class or Derived class.
Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.



Polymorphism

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Object Oriented Analysis and Design

Object oriented analysis performed for objects identification. After identifying the objects, the relationship among the objects should be identified and designed by using the concept OOD.

The purpose of OOAD is to

- Identify the object of a system.
- Identify their relationships
- Create the design, which can be converted to executable format using Object Oriented Language.

Object Oriented Analysis

Objectives of OOA are:

- To identify the classes.
- To identify the attributes of the classes
- To identify the methods required for each class
- To find out the class hierarchy.
- To find out the relationship and communication among the classes to represent the whole system.

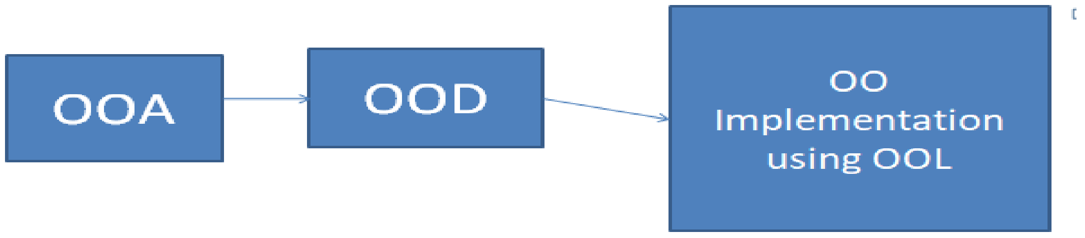
1. Structured analysis VS OOA

Structured analysis concentrate input– process—output requirements and function that process these data. Concentrate on individual components of the system. But OOA Class (function and data) is consider as the building block. The system behaviour, interaction among the classes that control the behavior of the whole system.

Structured analysis decompose the system in to top down approach.

OOA does not use top down decomposition to find out the classes present in the system.

There are 3 basic steps of applying and implementing OO concepts



Phase-1 : the aim of OOA it to Identify the objects in the system and describe the state and behavior of the objects , capacity of the object and communication among the object.

Phase-2: OOD, requirements should be realized. Establishment of association among the objects is complete the design.

Phase-3: Design can be converted in a running for using some object oriented programming languages. Such as C++ ,java etc.,

2. Domain Analysis

Objectives:

1. To understand the domain of the system belongs to.
2. To find out the reusable components already present in the domain.

Purpose of finding the reusable components are:

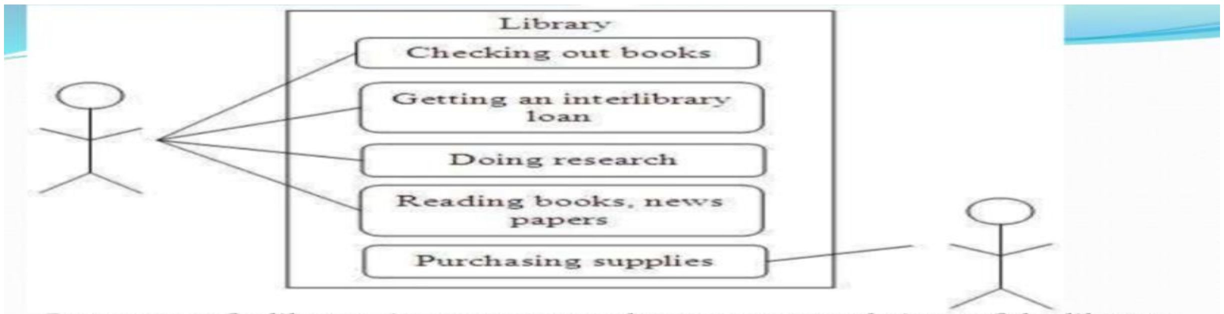
1. It reduce the development effort.
2. It reduce the total implementation time.
3. The standardization of the developed system is enhanced.

Domain Analysis Process:

- Define the domain: Identifies the domain of the system.
- Group the item in the domain: Similar components of the domain is grouped based on criteria such as implementation similarity, language used for implementation, functionality produced.
- Create the analysis model: use the reusable components to create the analysis model.
- Identify the reusability : after analysis model is created, evaluate the model to how much of the system component can be reused by the other system development.

Use Case Modeling

- The pictorial representation of the captured system requirement is called use case. A use case is an interaction between the users and system.
- The use case description must contains the following constrains:
- How the use case begins and end?
- When the use case begins and end?
- Interaction between the use cases (when the interaction occurs and what is exchanged).
- How and when the use case needs the data stored in the system.



Some uses of a library. As you can see, these are external views of the library system from the actor such as a member. The simpler the use case, the more effective it will be. It is unwise to capture all of the details right at the start; you can do that later.

OBJECT ORIENTED DESIGN

After identifies user requirements, the design process is started using UML.

The followings are important to OO design to be successful.

- Identifies the object and classes.
- Identifies the relationships between classes.
- Build the hierarchy for maximum reusability.
- Identifies the communication (message) among the classes.

1. OOD Modeling Techniques

Modeling techniques provides a set of rules and guidelines for analytics and designers to performing robust requirement analysis and design. It helps to modeling the business problem and implementation.

Some powerful modeling techniques are:

1.1. Rambaugh at al Object oriented modeling

1.2. Booch methodology

1.3. Jacobson's model

1.3. Jacobson's model

It comprises Object oriented business engineering, object oriented software engineering and object factory for software development.

These three methodologies together covers the entire software development life cycle phases such as requirement gathering, analysis, designing , coding and implementation. The following model s based on the use case model which captures the business requirement and evaluate into a object factory for checking reusability of the model.

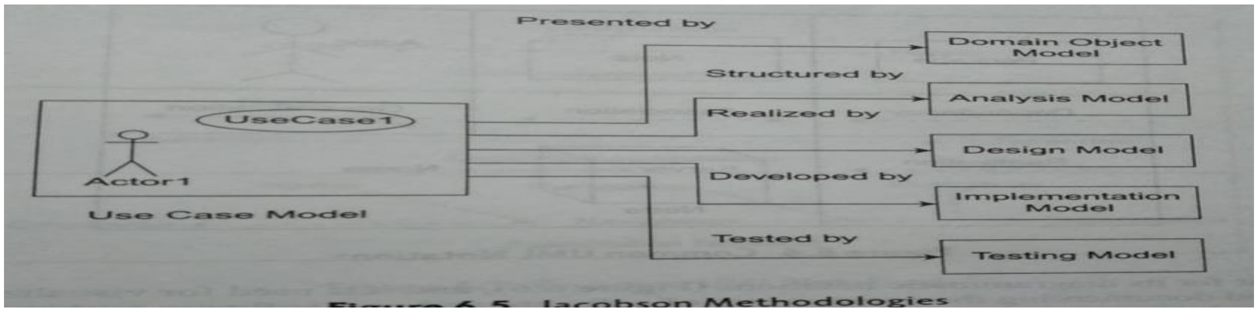


Figure 6.5 Jacobson Methodologies

User Interface Design

User interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, ships etc.

User interface is part of software and is designed such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction.

UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interfacing screens

UI is broadly divided into two categories:

- Command Line Interface
- Graphical User Interface

Command Line Interface (CLI)

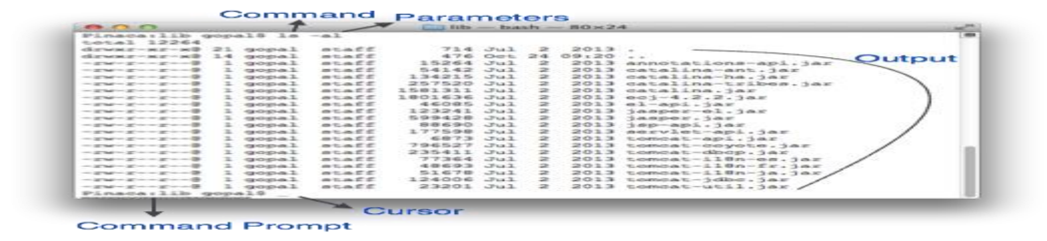
CLI has been a great tool of interaction with computers until the video display monitors came into existence. CLI is first choice of many technical users and programmers. CLI is minimum interface a software can provide to its users.

CLI provides a command prompt, the place where the user types the command and feeds to the system. The user needs to remember the syntax of command and its use. Earlier CLI were not programmed to handle the user errors effectively.

A command is a text-based reference to set of instructions, which are expected to be executed by the system. There are methods like macros, scripts that make it easy for the user to operate.

CLI uses less amount of computer resource as compared to GUI.

CLI Elements



```
Pinaca:lib gopal$ ls -al
total 12264
drwxr-xr-x@ 21 gopal  staff      714 Jul  2  2013 .
drwxr-xr-x@ 14 gopal  staff      476 Oct 24 09:20 ..
-rw-r--r--@  1 gopal  staff    15264 Jul  2  2013 annotations-api.jar
-rw-r--r--@  1 gopal  staff    54142 Jul  2  2013 catalina-ant.jar
-rw-r--r--@  1 gopal  staff   134215 Jul  2  2013 catalina-ha.jar
-rw-r--r--@  1 gopal  staff   257520 Jul  2  2013 catalina-tribes.jar
-rw-r--r--@  1 gopal  staff  1581311 Jul  2  2013 catalina.jar
-rw-r--r--@  1 gopal  staff  1801636 Jul  2  2013 ecj-4.2.2.jar
-rw-r--r--@  1 gopal  staff    46085 Jul  2  2013 el-api.jar
-rw-r--r--@  1 gopal  staff   123241 Jul  2  2013 jasper-el.jar
-rw-r--r--@  1 gopal  staff   599428 Jul  2  2013 jasper.jar
-rw-r--r--@  1 gopal  staff    88690 Jul  2  2013 jsp-api.jar
-rw-r--r--@  1 gopal  staff   177598 Jul  2  2013 servlet-api.jar
-rw-r--r--@  1 gopal  staff     6873 Jul  2  2013 tomcat-api.jar
-rw-r--r--@  1 gopal  staff   796527 Jul  2  2013 tomcat-coyote.jar
-rw-r--r--@  1 gopal  staff   235411 Jul  2  2013 tomcat-dbcp.jar
-rw-r--r--@  1 gopal  staff    77364 Jul  2  2013 tomcat-i18n-es.jar
-rw-r--r--@  1 gopal  staff    48693 Jul  2  2013 tomcat-i18n-fr.jar
-rw-r--r--@  1 gopal  staff    51678 Jul  2  2013 tomcat-i18n-ja.jar
-rw-r--r--@  1 gopal  staff   124006 Jul  2  2013 tomcat-jdbc.jar
-rw-r--r--@  1 gopal  staff    23201 Jul  2  2013 tomcat-util.jar
Pinaca:lib gopal$ _
```

A text-based command line interface can have the following elements:

- **Command Prompt** - It is text-based notifier that is mostly shows the context in which the user is working. It is generated by the software system.
- **Cursor** - It is a small horizontal line or a vertical bar of the height of line, to represent position of character while typing. Cursor is mostly found in blinking state. It moves as the user writes or deletes something.

- **Command** - A command is an executable instruction. It may have one or more parameters. Output on command execution is shown inline on the screen. When output is produced, command prompt is displayed on the next line.

Graphical User Interface

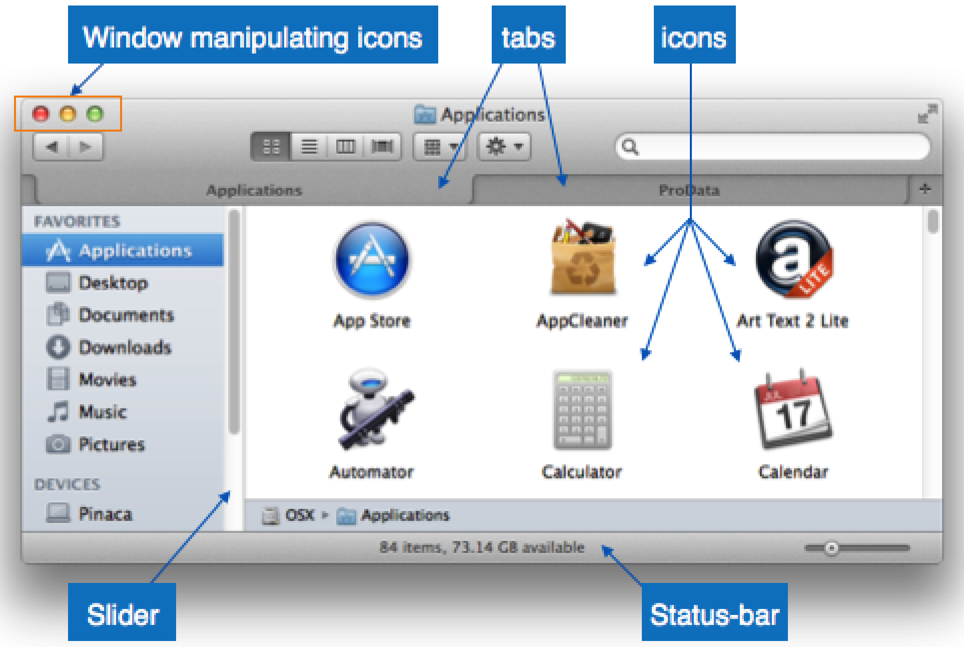
Graphical User Interface provides the user graphical means to interact with the system. GUI can be combination of both hardware and software. Using GUI, user interprets the software.

Typically, GUI is more resource consuming than that of CLI. With advancing technology, the programmers and designers create complex GUI designs that work with more efficiency, accuracy and speed.

GUI Elements

GUI provides a set of components to interact with software or hardware.

Every graphical component provides a way to work with the system. A GUI system has following elements such as:



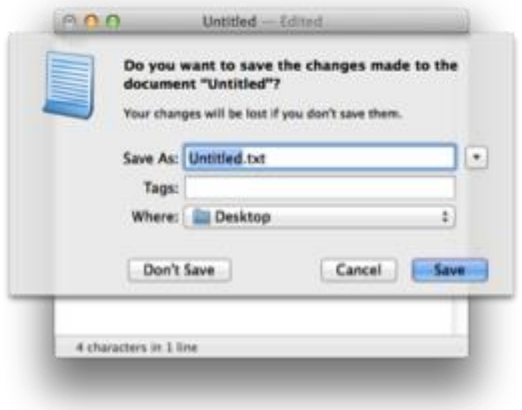
- **Window** - An area where contents of application are displayed. Contents in a window can be displayed in the form of icons or lists, if the window represents file structure. It is easier for a user to navigate in the file system in an exploring window. Windows can be minimized, resized or maximized to the size of screen. They can be moved anywhere on the screen. A window may contain another window of the same application, called child window.

- **Tabs** - If an application allows executing multiple instances of itself, they appear on the screen as separate windows. **Tabbed Document Interface** has come up to open multiple documents in the same window. This interface also helps in viewing preference panel in application. All modern web-browsers use this feature.
- **Menu** - Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.
- **Icon** - An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small pictures.
- **Cursor** - Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursors. On screen cursor follows the instructions from hardware in almost real-time. Cursors are also named pointers in GUI systems. They are used to select menus, windows and other application features.

Application specific GUI components

A GUI of an application contains one or more of the listed GUI elements:

- **Application Window** - Most application windows uses the constructs supplied by operating systems but many use their own customer created windows to contain the contents of application.
- **Dialogue Box** - It is a child window that contains message for the user and request for some action to be taken. For Example: Application generate a dialogue to get confirmation from user to delete a file.



Untitled — Edited

Do you want to save the changes made to the document "Untitled"?

Your changes will be lost if you don't save them.

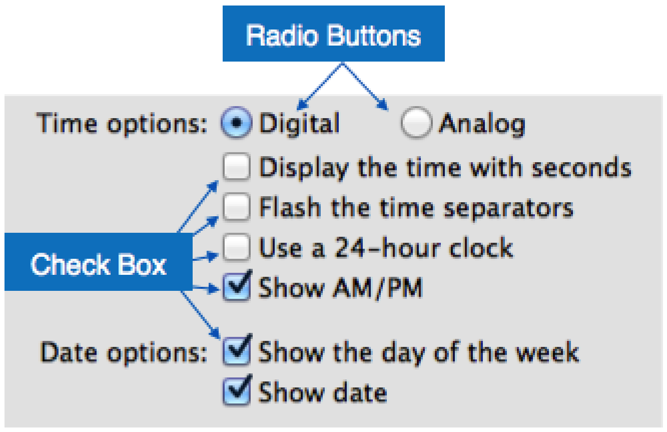
Save As: Untitled.txt

Tags:

Where: Desktop

4 characters in 1 line

- **Text-Box** - Provides an area for user to type and enter text-based data.
- **Buttons** - They imitate real life buttons and are used to submit inputs to the software.



Time options: [✓ Digital] Analog

Display the time with seconds

Flash the time separators

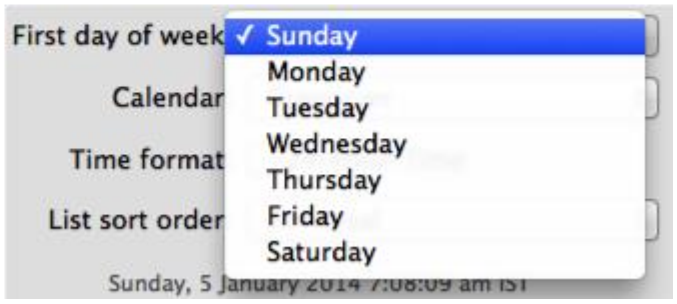
Use a 24-hour clock

[✓ Show AM/PM]

Date options: [✓ Show the day of the week]

[✓ Show date]

- **Radio-button** - Displays available options for selection. Only one can be selected among all offered.
- **Check-box** - Functions similar to list-box. When an option is selected, the box is marked as checked. Multiple options represented by check boxes can be selected.
- **List-box** - Provides list of available items for selection. More than one item can be selected.



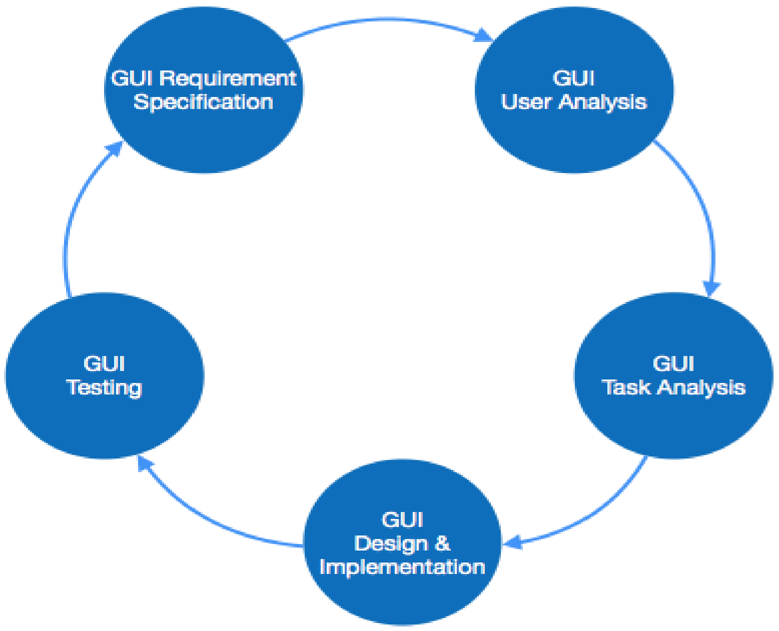
Other impressive GUI components are:

- Sliders
- Combo-box
- Data-grid
- Drop-down list

User Interface Design Activities

There are a number of activities performed for designing user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation among Waterfall, Iterative or Spiral Model.

A model used for GUI design and development should fulfill these GUI specific steps.



- **GUI Requirement Gathering** - The designers may like to have list of all functional and non-functional requirements of GUI. This can be taken from user and their existing software solution.
- **User Analysis** - The designer studies who is going to use the software GUI. The target audience matters as the design details change according to the knowledge and competency level of the user. If user is technical savvy, advanced and complex GUI can be incorporated. For a novice user, more information is included on how-to of software.
- **Task Analysis** - Designers have to analyze what task is to be done by the software solution. Here in GUI, it does not matter how it will be done. Tasks can be represented in hierarchical manner taking one major task and dividing it further into smaller sub-tasks. Tasks provide goals for GUI presentation. Flow of information among sub-tasks determines the flow of GUI contents in the software.
- **GUI Design & implementation** - Designers after having information about requirements, tasks and user environment, design the GUI and implements into code and embed the GUI with working or dummy software in the background. It is then self-tested by the developers.
- **Testing** - GUI testing can be done in various ways. Organization can have in-house inspection, direct involvement of users and release of beta version are few of them. Testing may include usability, compatibility, user acceptance etc.

GUI Implementation Tools

There are several tools available using which the designers can create entire GUI on a mouse click. Some tools can be embedded into the software environment (IDE).

GUI implementation tools provide powerful array of GUI controls. For software customization, designers can change the code accordingly.

There are different segments of GUI tools according to their different use and platform.

Example

Mobile GUI, Computer GUI, Touch-Screen GUI etc. Here is a list of few tools which come handy to build GUI:

- FLUID
- AppInventor (Android)
- LucidChart
- Wavemaker
- Visual Studio

User Interface Golden rules

The following rules are mentioned to be the golden rules for GUI design, described by Shneiderman and Plaisant in their book (Designing the User Interface).

- **Strive for consistency** - Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employed throughout.
- **Enable frequent users to use short-cuts** - The user's desire to reduce the number of interactions increases with the frequency of use. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.
- **Offer informative feedback** - For every operator action, there should be some system feedback. For frequent and minor actions, the response must be modest, while for infrequent and major actions, the response must be more substantial.
- **Design dialog to yield closure** - Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and this indicates that the way ahead is clear to prepare for the next group of actions.
- **Offer simple error handling** - As much as possible, design the system so the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple, comprehensible mechanisms for handling the error.

- **Permit easy reversal of actions** - This feature relieves anxiety, since the user knows that errors can be undone. Easy reversal of actions encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.
- **Support internal locus of control** - Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.
- **Reduce short-term memory load** - The limitation of human information processing in short-term memory requires the displays to be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.

UNIT-IV
Software Coding

Introduction

Software coding is the core aspect of software engineering that act as a catalyst for creating the software product.

1.Programming principles

- Computer programming is directly associated with writing a standard code.
- Computer programming means writing a standard code, testing the written code, and debugging and maintaining the code.

The five general programming principles :

1.Validity : The program must give the correct result which is valid.

2.Consistency: The program must do repeatedly what it intends to do. The program should give the output consistently.

3.Maintainablity: The program must be easily changeable and should have proper documentations.

4.Readability : The program must be easily readable so that it is easily maintainable.

5.Usability: The program must be usable for the specific purpose without any trouble.

2.Programming guidelines

- Guidelines means best practices. Programming best practices includes programming practices, naming conventions, comments, and programming principles rule of thumb.

3.Coding conventions (programming practices)

- Coding conventions are a set of best practices that helps to write the software code in an efficient manner.
- Advantages of coding convention:

  1.code becomes reusable saving money for all.
  2.it is easy to maintain.
  3. Enhancing the code becomes easy.

1.Naming convention

- Naming convention is a set of rules for choosing the character sequence to be used for identifiers (names).

  Name convention can be applied to:

  1.File name
  2.Folder name
  3.Variable name
  4.Function name
  5.Length of the identifiers
  6.Letter case and numerals
  7.Multiple word identifiers
  8.Hybrid conventions

2.Programming principles and rule of thumb

- Senior software engineers in various IT organizations do follow various programming principles and rules of thumb which are divided based on their experience and skills.
  - Choose sufficient data type
  - Try to avoid all variable as global variable

- Keep specific name for a variable
- Keep the variable and method names for one and one purpose
- Avoid writing public class for security purpose
- Avoid using database connection using specific users credentials.
- Avoid using forced data conversion

.Comments:

- Properly commented code serves no purpose for the compilation as well as executing the code, but it improves the readability of the code.
    - Keep the comments all ways up to date.
    - Write the comments before changing the actual code.
    - Try to avoid end of line comments
    - Use complete sentences
    - Remove un necessary comments.

4.White space best practices

- Blank lines and white spaces improve the readability of the code. It logically sub-divides the codes.
- Blank lines
- Blank spaces

Key concepts in software coding

1.Structured programming

- It is the subset of procedural programming. It is also called as modular programming.
- It represents the usage of a logical structure on the program being written to make it more readable, efficient, reliable, and easily maintained.
- Certain programming languages such as, pascal, COBOL, C, C++, Java and dBASE are designed with features that enforce a logical program structure.

2.Information Hiding

- Encapsulation uses the principle of information hiding.
- For example, in the class car engine, there are many parts inside it.
- Information hiding technique reduces complexity and increases usability and maintainability.

3.Coding standards and guidelines

- Good developers always follow the coding standards and guidelines while writing the code.
- Write comments before writing the code
- Use meaningful names for variables and functions
- Avoid using long name
- Use tool to improve coding standards
- Avoid deeply nested code
- Never use goto statement
- Do not duplicate the code
- Avoid using multiple inheritance.

Advantages of coding standards

- Reusability
- Maintainability
- Readability

    - Understandability
    - Robustness
    - Reliability

<u>4.Top-Down and Bottom-up coding</u>

Top-down coding: The top main module is written first before writing the bottom sub-modules. Bottom-up coding: It is the reverse of top-down coding, and here the implementation starts with the bottom sub-modules.

<u>5.Incremental Development of code</u>

- Project delivery is divided into parts- analyze, design, code, test, and deliver product.
- There are four phases:
- Inception :- requirement and scope
- Elaboration:- non functional requirements
- Construction:- production
- Transition:- deployment of the code into production

<u>6.Programming style</u>

- Set of rules and guidelines followed while writing a computer program is called as programming style.

<u>7.Code sharing</u>

- Code share is famous in aviation business arrangement where one flight is shared by different airline operators.

<u>8.Code Review</u>

- Code review is a systematic process, which is also called as peer review which is used to find and fix the mistakes in the early phase of the coding and it helps to improve the code quality and improve the coding skills of the developers.
    - Formal inspections
    - Lightweight code review
    - Pair programming
    - Automatic code review software.

<u>9.Static program Analysis</u>

- Static program analysis is the analysis of computer software that is performed without actually executing programs.
    - Unreachable code
    - Undeclared variable
    - Unused variable
    - Unused functions
    - Parameter type mismatches.

<u>10.Symbolic execution</u>

- Symbolic execution refers to the analysis of programs by tracking symbolic value rather than actual values.

11. Code inspections:

Code Inspection is the most formal type of review, which is a kind of static testing to avoid the defect multiplication at a later stage.

- The main purpose of code inspection is to find defects and it can also spot any process improvement if any.
- An inspection report lists the findings, which include metrics that can be used to aid improvements to the process as well as correcting defects in the document under review.
- Preparation before the meeting is essential, which includes reading of any source documents to ensure consistency.
- Inspections are often led by a trained moderator, who is not the author of the code.
- The inspection process is the most formal type of review based on rules and checklists and makes use of entry and exit criteria.
- It usually involves peer examination of the code and each one has a defined set of roles.
- After the meeting, a formal follow-up process is used to ensure that corrective action is completed in a timely manner.

12. Rapid prototyping:

Rapid prototyping the prototype is being created automatically using CAD. It requires 3 to 72 hours building in object which saves a lot of time .

- Create the design of the product using CAD.
- Convert the CAD model into STL form
- Slice STL format file into cross sectional layers
- Convert each layer into a model
- Finish and tune the model
- Programming

13. Intelligent software agent:

Software act as a agent of a user which converts and reports set of activities can decide the course of action to be taken on behalf of users based on set of predefined set of codes.

Ex:

- Virus scanning software
- Data warehousing agents
- Automatic share trader

14.Software Reuse

- Reusability is the ability to reuse parts of code to create new code. Reusable part includes entire code, functions, classes, and data structures.
    - Consumer
    - Contributor
    - Facilitator

Introduction to software measurement and metrics

Measurement

- It is a process or the result by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them in order to clearly defined rules.

  Software measurement: It is a quantified attributes of a characteristic of a software product or the software process.

1.Why and where measurements in important in life

Measurement is not only important for software, but also in our day-to-day life also.

- To take proper medicine
- To cook correctly
- To pay correct money at shop
- To come office on time
- To measure performance of the students.

2.Why to measure software?

- To understand what we do
- For realistic planning
- For control
- To improve the process and product
- Cost reduction
- To define quality level

3.Characteristics of effective measurement program.

- Aligned with the organization business objectives
- Tied to decision making at lowest level possible
- Focused on measuring processes
- Viewed as mission critical.

4. Types of Measure:

- Direct measure: It is measure directly in terms of the observed attribute.
- Indirect measure: It is calculated from other direct and indirect measures.
- True performance measures: Customers use true performance measures.

5. Activities of a measurement process

- Plan, Collect, Analyze, Interpret, Circulate, Feedback, Improvement.

Metrics

Metrics is a standard of measurement. A metric indicates the nature and/or strength of an attribute, usually by means of a number, often in conjunction with a unit.

1.GQM paradigm of metrics (Goal, Question, Metric):

- Establish the measurement goal that is specific to the process or product under study.
- Define a set of question that are appropriate.
- Identify metrics that help to answer the questions identified in step2.

2.Black box and White box metrics:

- Metrics are also classified as black box metrics and white box metrics.
- If the inside and outside logic of a metrics is not known then it is called as Block Box Metric.
- If the inside and outside logic of a metrics is clearly known then it is called as White Box Metric.

3.Types of software metrics (classification of metrics):

- Product metrics: metrics that are related to the product (LOC)
- Process metrics: metrics that are related to the process of development (length of the iteration)
- Resources metrics: metrics that are related to the usage of personnel and resources and properties (developer productivity)

- External metrics: measured with respect to environment (productivity, usability)
- Internal metrics: not dependent on the environment (LOC)
- Direct metrics: directly measurable (LOC, FP)
- Indirect metrics: not measurable directly (code quality)

Software configuration management

- SCM is the overall management of a software design project, as it evolves into a software product or system.

1.Basics concepts of configuration management

- Configuration item-> A piece of software or work product which is subject to change is a configuration item.
- Version and configurations-> A version identifiers the state of a particular configuration item.
- Promotion-> A promotion is a version of a configuration item.
- Release-> A release is a version that is available to the user or the client.
- Repository-> A repository stores various releases of a CM item.

Software configuration management process

- The traditional SCM identifies four procedures that must be defined for each software project to ensure that a good SCM process is implemented, which are as follows:
- Configuration Identification
- Configuration control
- Configuration status accounting
- Configuration authentication

1.Configuration Identification

- Any software is usually made up of several programs. The following attributes:
- Functionality complete-> A baseline exhibits a defined functionality.
- Known quality-> The quality of a baseline is well-defined.
- Immutable and completely re-creatable-> A baseline, once-defined, cannot be changed.

2.Configuration control

- Configuration control is the process of deciding, coordinating the approved changes for the proposed CI and implementing the changes on an appropriate baseline.

3. Configuration status accounting

- Configuration status accounting keeps a record o all the changes made to the previous baseline to reach the new baseline.

4. Configuration Authentication

- Configuration authentication is the process of assuring that all the planned and approved changes have been incorporated in new baseline.

Tools used in software configuration management

- Free software tools that help in SCM are
    - 1.Concurrent Versions System (CVS)
    - 2. Revision Control System (RCS)
    - 3. Source Code Control System (SCCS)

- Concurrent versions system-> it is a version control system. It is used to record the history of the source file.
- Revision control system-> in order to change a configuration item, the developer has to look that item first, and prevent other developers from changing the item at the same time.
- Source code control system-> it is a complete set of commands that allow specified users to control and track changes made to an SCCS file.

Software Configuration Management Plan(SCMP)

is a document that describe way the configuration management will be implemented and followed in an organization.

- Software configuration management Resources
- Project Manager
- Configuration Management Manager(CMM)
- configuration control board(CCD)
- Software configuration management tool
- Change request and response
- Change control process
- Management of resource documentation.

Project Management Introduction

Process:

- A process is a set of interrelated activities performed to create pre-specified products, services, or results.

Project: The PMI defines a project as a temporary endeavor undertaken to create a unique product, service or result.

- There are three key words that define the term project.

1.Temporary
2.Unique
3.Progressive Elaboration

Project management

Project management involves application of knowledge, skills, tools, and techniques to control project activities so as to meet project requirement.

The project management life cycle has five distinct phases:

1. Initiation
2. Planning
3. Execution
4. Control
5. Closure

Software Maintenance

- Software maintenance is one of the core aspects of software engineering.
- The process of modifying the production system after the delivery to correct the faults, for improving performance, and for adapting to the changing environment is called as software maintenance.

1.Maintenance Activities

- There are four types of maintenance activities namely,
  1. Adaptive maintenance
  2. Corrective maintenance
  3. Perfective maintenance
  4. Preventive maintenance

1.Adaptive maintenance

- Adjusting the system adaptively based on the environment changes is called as adaptive maintenance.

2.Corrective maintenance

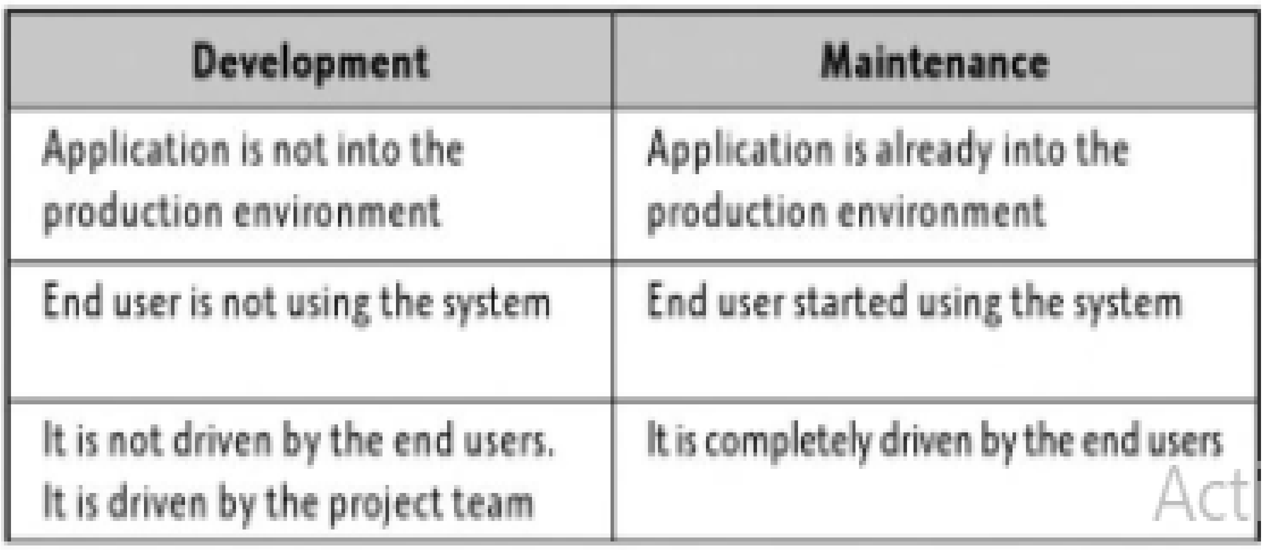
- It is the process of identifying, isolating, and repairing the faults that are discovered in the system so that the system can be restored and operational.

3.Perfective maintenance

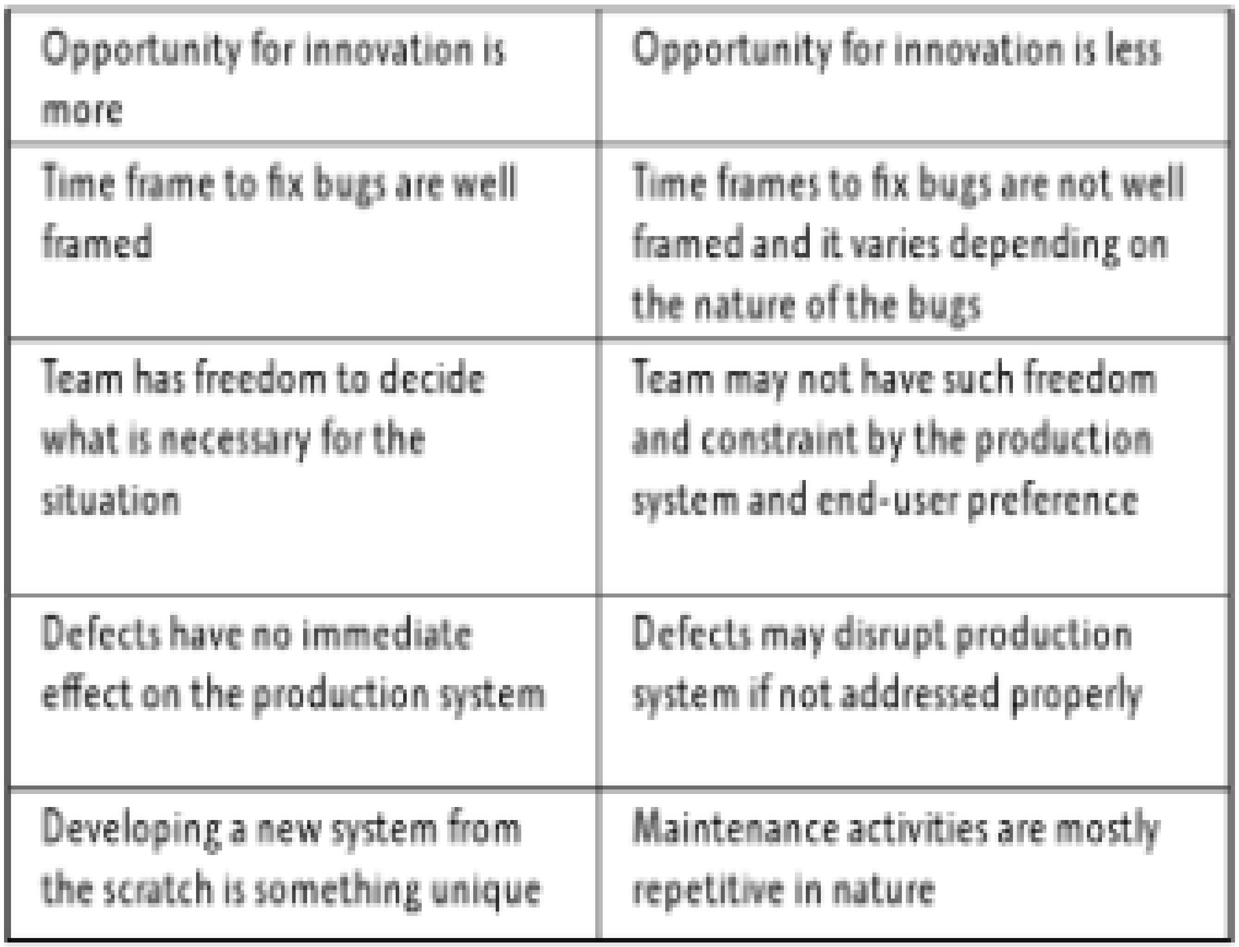
- Perfective maintenance happens due to adapting changes and due to user requirements.

4.Preventive maintenance

- Activities that are aimed at increasing the maintainability are called as preventive maintenance.



Development	Maintenance
Application is not into the production environment	Application is already into the production environment
End user is not using the system	End user started using the system
It is not driven by the end users. It is driven by the project team	It is completely driven by the end users



Opportunity for innovation is more	Opportunity for innovation is less
Time frame to fix bugs are well framed	Time frames to fix bugs are not well framed and it varies depending on the nature of the bugs
Team has freedom to decide what is necessary for the situation	Team may not have such freedom and constraint by the production system and end-user preference
Defects have no immediate effect on the production system	Defects may disrupt production system if not addressed properly
Developing a new system from the scratch is something unique	Maintenance activities are mostly repetitive in nature

<u>Introduction to software testing</u>

<u>1.Psychology testing:</u>

Over the years as software testing has evolved from being synonymous with debugging to a separate independent competency aimed at improving the overall quality of the product.

<u>Testing evolution process</u>

There are five phases:

- Rudimentary level phase 0 : In the very beginning, the aim is to get a bug-free product without any coding errors.
- Basic level-phase I: It is to show that the software works.
- Intermediate level-phase II : It is to find areas where the software does not work.
- Advanced level Phase III : The advanced level phase III of thinking evolved based on the level of confidence the tester develops on the software he is testing.
- Optimized level Phase IV : The optimized level phase IV of thinking is based on a strong expertise in the areas to be tested and identification of what can be tested or not

<u>2.Software testing scope</u>

- These can be broadly classified into two main areas namely, applications software testing and systems software testing.

- Application software testing: It is testing the common applications which is seen every day and mostly used for data processing, such as banking system, payroll, inventory management system.
- Systems software testing: It is done on the programs written for running the complete/ system itself which facilitate, enhance, and control various programs such as the operating system, compiler and interpreter.

<u>3.Software testing objectives</u>

- The first objective of software testing is to prevent bugs from getting into the system. This type of testing is commonly called as verification testing.

<u>1.Verification Testing:</u>

- Typically a tester may be asked to prepare a set of test cases or execute and may not be familiar or experienced with the verification techniques.

  <u>Following techniques helps for verification:</u>
  - Reviews
  - Walkthrough
  - Inspection

<u>2.Validation testing:</u>

- The testing activities that are done to evaluate the software in the executable mode can be called as validation testing.

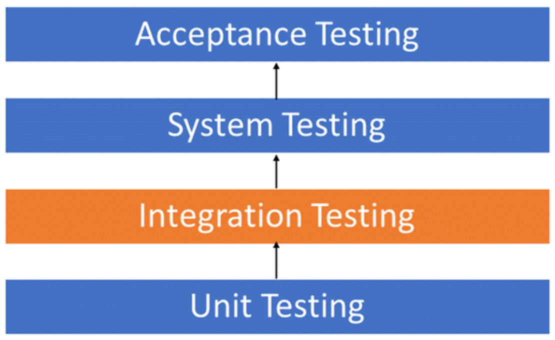
  <u>Some of the issues in validation</u>
  - Lack of time
  - Software crashes
  - Missing critical defects

  <u>Validation testing basic phases</u>
  - Unit testing-> The smallest testable units of software which could be individual programs or modules are tested during the unit testing phase.

INTEGRATION TESTING

Integration testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.

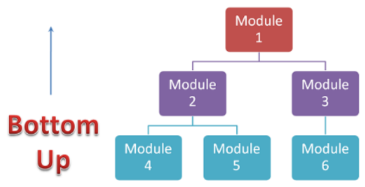
Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as **'I & T'** (Integration and Testing), **'String Testing'** and sometimes **'Thread Testing'**.



<u>Bottom-up Integration Testing</u>

Bottom-up Integration Testing is a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.

Diagrammatic Representation:



Advantages:

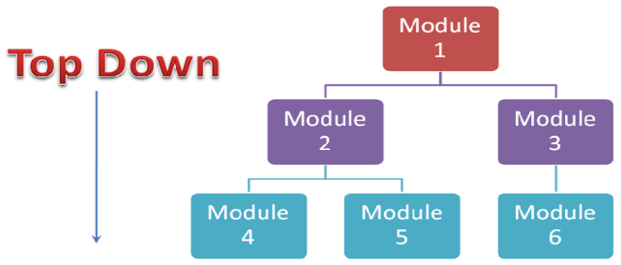
- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible

<u>Top Down Integration Testing</u>

Top-down Integration Testingis a method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.



Advantages:

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.
- Functional testing-> To verify the end-to-end functionalities of the modules.

System Testing

Where all the components of the system software, hardware, external interfaces are all tested as per the specifications.

System Testing includes testing of a fully integrated software system. Generally, a computer system is made with the integration of software (any software is only a single element of a computer system). The software is developed in units and then interfaced with other software and hardware to create a complete computer system.

In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware. System testing is a series of different type of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.

It is **end-to-end testing** where the testing environment is similar to the production environment. **System testing falls under Black box testing** as it includes testing of the external working of the software. Testing follows user's perspective to identify minor defects.

System Testing includes the following steps.

- Verification of input functions of the application to test whether it is producing the expected output or not.
- Testing of integrated software by including external peripherals to check the interaction of various components with each other.
- Testing of the whole system for End to End testing.
- Behavior testing of the application via auser's experience

User Acceptance Testing (UAT)

This type of Acceptance Testing, also known as Beta Testing, is performed by the end users (either existing or potential) of the software. They can be the customers themselves or the customers' customers or the general public.

Acceptance Criteria

Acceptance criteria are defined on the basis of the following attributes

- Functional Correctness and Completeness
- Data Integrity
- Data Conversion
- Usability
- Performance
- Timeliness
- Confidentiality and Availability
- Installability and Upgradability
- Scalability
- Documentation

Types of Software testing

- The basic categories of testing namely, white and black box 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.

Static Testing is a software testing technique which is used to check defects in software application without executing the code. Static testing is done to avoid errors at an early stage of development as it is easier to identify the errors and solve the errors. It also helps finding errors that may not be found by Dynamic Testing.

Structural testing, also known as glass box testing or **white box testing** is an approach where the tests are derived from the knowledge of the software's structure or internal implementation. The other names of structural testing includes clear box testing, open box testing, logic driven testing or path driven testing.

Structural 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 are covered.

Advantages of Structural Testing:

- Forces test developer to reason carefully about implementation
- Reveals errors in "hidden" code
- Spots the Dead Code or other issues with respect to best programming practices

Disadvantages of Structural Box Testing:

- Expensive as one has to spend both time and money to perform white box testing.
- Every possibility that few lines of code is missed accidentally.
- In-depth knowledge about the programming language is necessary to perform white box testing.

Black box Testing?

Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as Specifications based testing. Independent Testing Team usually performs this type of testing during the software testing life cycle.

Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.

- **Functional testing** - This black box testing type is related to the functional requirements of a system; it is done by software testers.

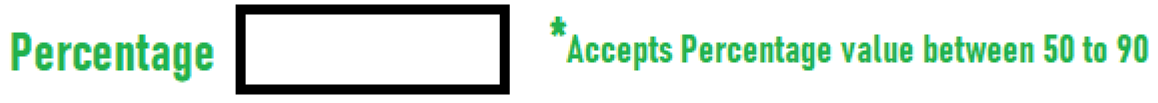
- **Non-functional testing** - This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** - Regression Testing is done after code fixes, upgrades or any other system maintenance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

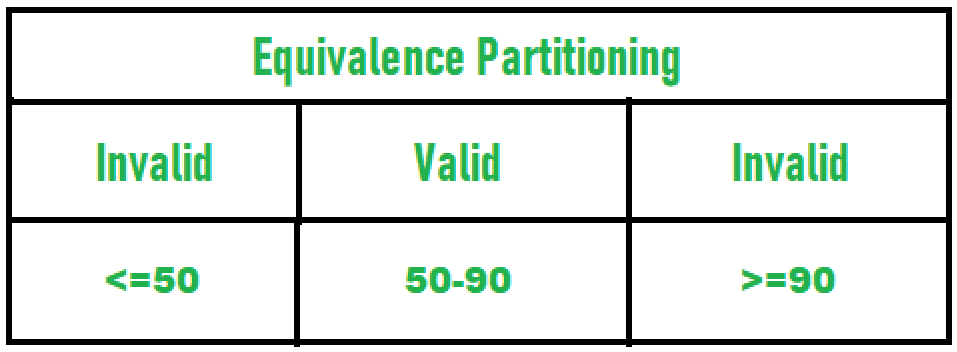
<u>Black box Testing Techniques:</u>
There are different techniques involved in Black Box testing.

- Equivalence Class
- Boundary Value Analysis
- Domain Tests
- Decision Tables

Equivalence Class Testing: It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.

Example-1:
Let us consider an example of any college admission process. There is a college that gives admissions to students based upon their percentage.
Consider percentage field that will accept percentage only between 50 to 90 %, more and even less than not be accepted, and application will redirect user to an error page. If percentage entered by user is less than 50 %or more than 90 %, that equivalence partitioning method will show an invalid percentage. If percentage entered is between 50 to 90 %, then equivalence partitioning method will show valid percentage.



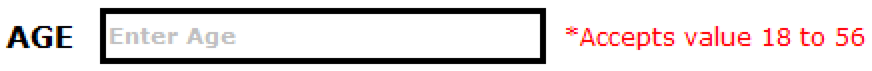


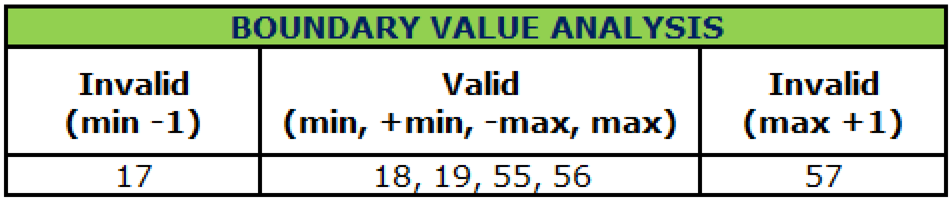
Equivalence Partitioning		
Invalid	Valid	Invalid
<=50	50-90	>=90

Boundary Value Testing: Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges.

Example on Boundary Value Analysis Test Case Design Technique:

Assume, we have to test a field which accepts Age 18 – 56

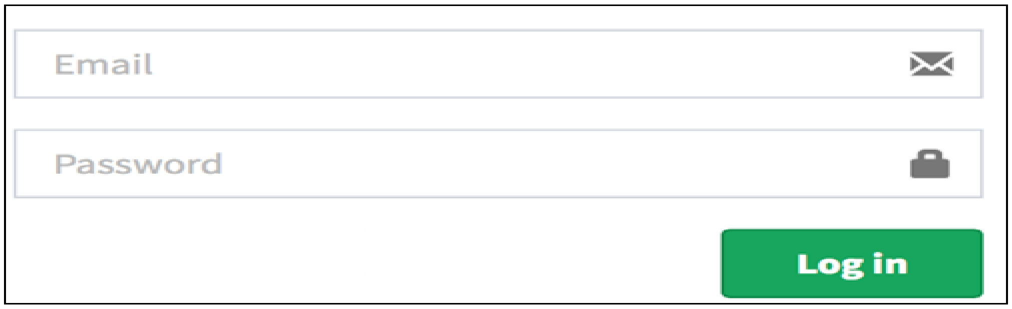




BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	Valid (min, +min, -max, max)	Invalid (max +1)
17	18, 19, 55, 56	57

Decision Table Testing: A decision table puts causes and their effects in a matrix. There is a unique combination in each column.
Decision table testing is a software testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form. That is why it is also called as a **Cause-Effect** table where Cause and effects are captured for better test coverage.

Example 1: How to make Decision Base Table for Login Screen
Let's create a decision table for a login screen.



The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

State Transition Testing: This testing technique uses the inputs, outputs, and the state of the system during the testing phase. It checks the software against the sequence of transitions or events among the test data.

Based on the type of software that is tested, it checks for the behavioral changes of a system in a particular state or another state while maintaining the same inputs.

How to do Black Box testing

When you get the basic understanding of black-box testing then the next question which comes up in mind is: How to perform the Black box testing? Below you can check the steps to perform this testing:

- The first step to black-box testing is to understand the requirement specifications of the application under test. An accurate and precise SRS document should be there.
- The next step is to evaluate the set of valid inputs and test scenarios to test the software. The goal is to save time and get good test coverage.
- Prepare the test cases to cover a maximum range of inputs.
- The test cases are run in the system to generate output, which is validated with the expected outcome to mark pass or fail.
- The failed steps are marked and sent to the development team to fix them.
- Retest the system using various testing techniques to verify its recurring nature or to pass it.
- The black box testing can be easily used to check and validate the entire software development life cycle. It can be used at various stages such as unit, integration, acceptance, system, and regression to evaluate the product.

Why should we use Black Box Testing?

Tools of Black box testing are basically record and playback ones. These tools record the test cases in the form of scripts like TSL, JavaScript, VB script, etc. All of these tools are basically used for regression testing in order to check whether the provided new build has made any defect in already fine working application functionality.

Below are the advantages:

- Black box testers also do not need to know any programming languages.
- Black box tests are always executed from a user’s point of view since it would help in exposing discrepancies significantly
- Black box testers do not need to know how the software has been implemented.
- Test cases related to black box are designed by testers as soon as the specifications are in the completed stage.

When we do Black Box testing?

- Unlike traditional white box testing, black box testing is beneficial for testing software usability.
- The overall functionality of the system under test
- Black box testing gives you a broader picture of the software.
- This testing approach sees an application from a user’s perspective.
- To test the software as a whole system rather than different modules.

UNIT-V

Web engineering

1.Introduction to WEB

- Literal meaning of web is “a network of thread constructed by spider”.
- In software engineering, the term web is a collection of computers connected by a network.

1.General Web characteristics

- Client server is the basic concept under which world wide web system acts.
- Client machine or clients is the computer that uses the applications of the web and retrieves and uses data.
- Client machine has a software called as web browser.
- A web browser software which runs on a client machine helps to access the world wide web.
- Popular web browsers are:
    - Microsoft Internet Explorer
    - Google chrome web browser
- Server machine or server is the computer in which the actual applications and information are stored.

2.Web application categories

Web application are categorized into three based on where it runs:

1.Client-side web applications
2.Server-side web applications
3.Middleware web applications

1.Client-side web applications:- This web applications runs at the client side in software. Such as, HTML, and JAVA Scripts.

2.Server-side web applications:- This web applications runs at the server only. Such as ASP, PHP.

3.Middleware web applications:- This web applications acts in between the client and server. Such as, java beans, java servlet.

Usage of web application:

- Online news provider
- Online web e-mail provider
- Online chat sites
- Social networking sites
- Portals

Advantages of web applications

1. Applications can be accessed from anywhere in the world from any machine.
2. Only browser is needed at the client side to connect with the web application.
3. Any operating system will run the applications.
4.upgrading the web application is easy as it required the application to be updated only at a single place.

- Drawbacks of web applications
  - Security is the main drawback of any web applications
  - Data theft is possible easily in web application
  - Application also can be copied easily in the web.

WEB Engineering

Is the applications of systematic, disciplined and quantifiable approach to the design, production, operation, deployment, maintenance and evolutions of web.

2.Emerging trends in software engineering

1.Introduction:- Software engineering field is evolving constantly as well as rapidly, so adapting to new changes is imperative for success of this field. Such as Technology, business environment, economy situation.

2.Web 1.0:- is considered as web pages created with static pages (HTML) and is completely content-driven and just reading the website and it is just one-way communication.

Web 2.0 is considered as a collaborative technology, which is more than the static web pages and dynamic web pages. Users can also edit and update the website content.

- Web 2.0 uses social media for interactions. For example, gmail, Google maps.
- Web 2.0 websites include following features and techniques:
- Search, links, tags, extensions, signals.

3.RAPID delivery

- Software engineering principles are being followed in parallel rather than in sequence, as the customer wants to see the outcome quickly.
- A time box is fixed and whatever possible during this boxing period will be delivered as a workable incremental software product to the customer.
- Example of RAPID delivery:
- XP methodology
- Dynamic Software Development Methodology (DSDM).

4.Open Source Software Development

- Open source not only gives the executable code to the users but also the source code to the users, which has a lot of advantages.
- A few examples of open source software products,

1.Mozilla Firefox – famous web browser
2.word press – web publishing platform (website creation)
PHP – Scripting languages of web
4.Linux – famous operating system (like UNIX).

Security Engineering

- Security engineering is a specialized field of engineering that focuses on the security aspects of the software system.
- Security is given high priority during the requirement analysis and designing stage of the system itself.
- Default deny and default permit are two fundamental principles of security engineering aspects.
- Default deny indicates that the call to the source code is denied except, special permission calls.
- Default permit mode, everything is default permitted which is a big security threat to the code.

Service-Oriented Software Engineering

- Service-oriented software engineering is the development of software system using reusable components and services from other providers.
- SOSE acts as a middle layer between the business and system developers.

WEB Service

- Web service concepts are famous because of SOSE . Services usually do not have interfaces and are independent of each other.
- Services depend solely on the input messages that are being passed on to it, which are usually in the XML format, Web Service Description Languages (WSDL), which is also XML based, is used to describe the services provided, format of the service, accessibility of the service.
- Three types of players in service-oriented interaction:
    - Service providers
    - Service users
    - Service registries

Software as a service

- Software as a service (SAAS) is a model of software delivery, where the software company provides maintenance, daily technical operations, and support for the software provided to their client.

Key characteristics of SAAS:

- Software is available through internet
- Services charged on pay per user basis
- Upgradation of services happens to all at the same time.

Disadvantages of SAAS

- It fully depends on broadband connection
- Customization is little

Service-Oriented Architecture

- SOA does not provide full business as service, instead a small process.
- It is a manufacturing hub, where it allows application to exchange data with one another.

Cloud Computing

- IT infrastructure as a service is called as cloud computing.
- Cloud application may be delivered using SAAS.
- In cloud computing, infrastructure itself is programmable.

Basic component of client computing are

(1) clients,
(2) services,
(3) applications,
(4) platform,
(5) storage, and
(6) physical infrastructure.

Aspect-Oriented Software Development (AOSD)

- Aspect means concerns, concerns of the system are separately addressed as an object and are being addressed throughout the software life cycle.
- The entire system is modularized into objects based on functionality, and a special object called as Aspect is also created, which represent the aspects of the entire system.

Test-Driven Development (TDD)

- In this concept, a testing code is written separately in order to test a software code.
- The testing code is written even before the actual code is written.
- Now, the developer writes the basic skeleton of the code and then compiles the testing code, which will now get compiled.

Social computing

- Social computing means using the social software to connect with society.
- There are two main components of social computing:
  1) contents and
  2) Connections.
- Contents are getting shared through (by) connections, for example, face book, twitter, instant message (Skye, Google talk).
- Social computing is usually used for marketing, and communication purposes.