MAJOR BASED ELECTIVE I (A)
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

RequirmentsAnalysis 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, ChandramouliSeetharaman, B.G.Geetha, Pearson Publications, 2015

Reference Books:

1. Software Engineering, Jibitesh Mishra, Pearson Education, 2011

*****

SOFTWARE ENGINEERING

UNIT-I

.

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.

Definitions

IEEE defines software engineering as:

(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.

(2) The study of approaches as in the above statement.

Fritz Bauer, a German computer scientist, defines software engineering as:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

Software Process

A software process (also knows as software methodology) is a set of related activities that leads to the production of the software..

Any software process must include the following four activities:

1. **Software specification** (or requirements engineering): Define the main functionalities of the software and the constrains around them.
2. S**oftware design and implementation**: The software is to be designed and programmed.
3. **Software verification and validation**: The software must conforms to it's specification and meets the customer needs.
4. **Software evolution** (software maintenance): The software is being modified to meet customer and market requirements changes.

There are five generic process framework activities:

1. Communication:
The software development starts with the communication between customer and developer.

2. Planning:
It consists of complete estimation, scheduling for project development and tracking.

3. Modeling:

- Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.
- The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

4. Construction:

- Construction consists of code generation and the testing part.
- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not.
- Testing also check that the program provides desired output.

5. Deployment:

- Deployment step consists of delivering the product to the customer and take feedback from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software
- In software development life cycle, various models are designed and defined. These models are called as **Software Development Process Models.**
- On the basis of project motive, the software development process model is selected for development.

DIFFERENT SOFTWARE DEVELOPMENT PROCESS MODELS:

1) Big-Bang model
2) Code-and-fix model
3) Waterfall model
4) V model
5) Incremental model
6) RAD model
7) Agile model
8) Iterative model
9) Spiral model
10) Prototype model

1) Big-Bang Model

- Big-Bang is the SDLC(Software Development Life cycle) model in which no particular process is followed.
- Generally this model is used for small projects in which the development teams are small. It is specially useful in academic projects.
- This model is needs a little planning and does not follow formal development.
- The development of this model begins with the required money and efforts as an input.
- The output of this model is developed software, that may or may not be according to the requirements of the customer.

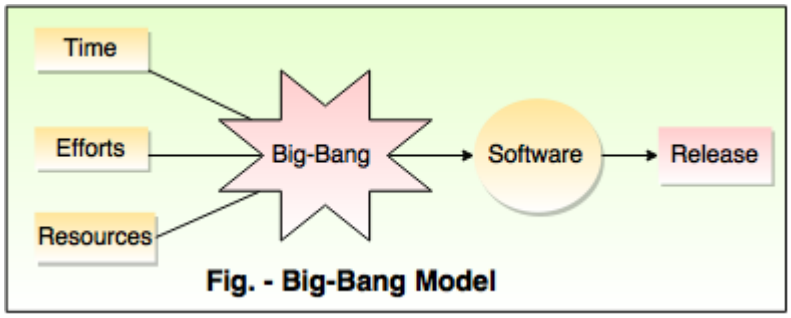


Fig. - Big-Bang Model

Advantages of Big-Bang model

- Big-Bang model is a simple model.
- It needs little planning.
- It is simple to manage. It needs just a few resources to be developed.
- It is useful for students and new comers.

Disadvantages of Big-Bang model

- It is a very high risk model.
- This model is not suitable for object oriented and complex projects.
- Big-Bang is poor model for lengthy and in-progress projects.

2) Code-and-fix Model

- Code and fix model is one step ahead from the Big-Bang model. It identifies the product that must be tested before release.
- The testing team find the bugs then sends the software back for fixing. To deliver the fixes developers complete some coding and send the software again for testing. This process is repeated till the bugs are found in it, at an acceptable level.

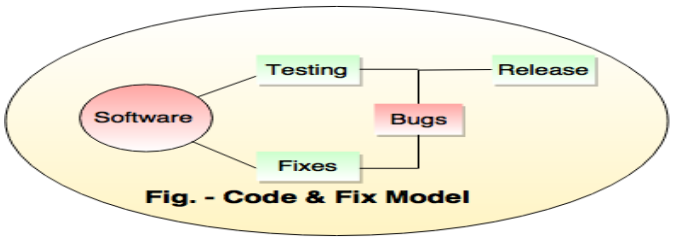


Fig. - Code & Fix Model

Advantages of Code-and-fix model

- This model is suitable for small projects.
- It needs less project planning.

Disadvantages of Code-and-fix model

- It is difficult to accommodate changes.
- It is not clear what will be delivered and when.
- It is difficult to assess quality.

3) Waterfall Model

- The waterfall model is the classic model or oldest model and is known as mother of all the model. It is widely used in government projects and many vital projects in company.
- The waterfall model is also called as **'Linear sequential model'** or **'Classic life cycle model'.**
- In this model, each phase is executed completely before the beginning of the next phase. Hence the phases do not overlap in waterfall model.
- This model is used for small projects.
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is completed.

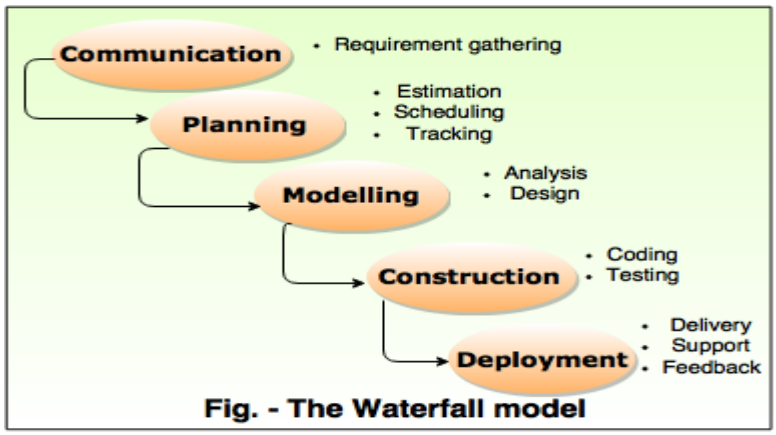


Fig. - The Waterfall model

Following are the phases in waterfall model:

i) Communication
The software development starts with the communication between customer and developer.

ii) Planning
It consists of complete estimation, scheduling for project development.

iii) Modeling

- Modeling consists of complete requirement analysis and the design of the project i.e algorithm, flowchart etc.
- The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

iv) Construction

- Construction consists of code generation and the testing part.
- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not.
- Testing also checks that the program provides desired output.

v) Deployment

- Deployment step consists of delivering the product to the customer and taking feedback from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

Advantages of Waterfall model

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

Disadvantages of the Waterfall model

- This model is not good for complex and object oriented projects.
- In this model, the changes are not permitted so it is not fit for moderate to high risk changes in project.
- It is a poor model for long duration projects.
- The problems with this model are uncovered, until the software testing.
- The amount of risk is high.

4) V Model

- V model is known as Verification and Validation model.
- This model is an extension of the waterfall model.
- In the life cycle of V-shaped model, processes are executed sequentially.
- Every phase completes its execution before the execution of next phase begins.

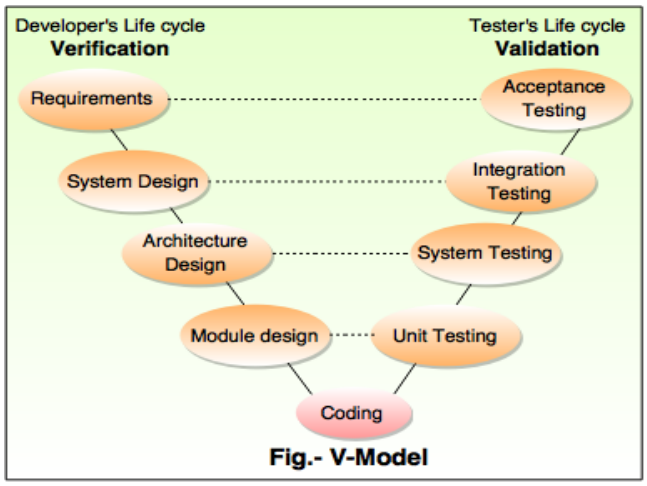


Fig.- V-Model

Following are the phases of V-model:

i) Requirements

- The requirements of product are understood from the customers point of view to know their exact requirement and expectation.
- The acceptance test design planning is completed at requirement stage because, business requirements are used as an input for acceptance testing.

ii) System Design

- In system design, high level design of the software is constructed.
- In this phase, we study how the requirements are implemented their technical use.

iii) Architecture design

- In architecture design, software architecture is created on the basis of high level design.
- The module relationship and dependencies of module, architectural diagrams, database tables, technology details are completed in this phase.

iv) Module design

- In module phase, we separately design every module or the software components.
- Finalize all the methods, classes, interfaces, data types etc.
- Unit tests are designed in module design phase based on the internal module designs.

- Unit tests are the vital part of any development process. They help to remove the maximum faults and errors at an early stage.

v) Coding Phase

- The actual code design of module designed in the design phase is grabbed in the coding phase.
- On the basis of system and architecture requirements, we decide the best suitable programming language.
- The coding is executed on the basis of coding guidelines and standards.

Advantages of V-model

- V-model is easy and simple to use.
- Many testing activities i.e planning, test design are executed in the starting, it saves more time.
- Calculation of errors is done at the starting of the project hence, less chances of error occurred at final phase of testing.
- This model is suitable for small projects where the requirements are easily understood.

Disadvantages of V-model

- V-model is not suitable for large and composite projects.
- If the requirements are not constant then this model is not acceptable.

Difference between the Verification and Validation

Verification	Validation
Verification is the process to find whether the software meets the specified requirements for particular phase.	The validation process checks whether the software meets requirements and expectations of the customer.
It evaluates an intermediate product.	It evaluates the final product.
The objective of verification is to check whether software is constructed according to requirement and design specification.	The objective of validation is to check whether the specifications are correct and satisfy the business need.
It describes whether the outputs are as per the inputs or not.	It explains whether outputs are accepted by the user or not.
Verification is completed before the validation.	It is completed after the verification.
Plans, requirement, specification, code are evaluated in the verifications.	Actual product or software is tested under validation.

5) Incremental Model

- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.

- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for suggesting any modifications.
- The next increment implements the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is completed.
  For example, the word-processing software is developed using the incremental model.

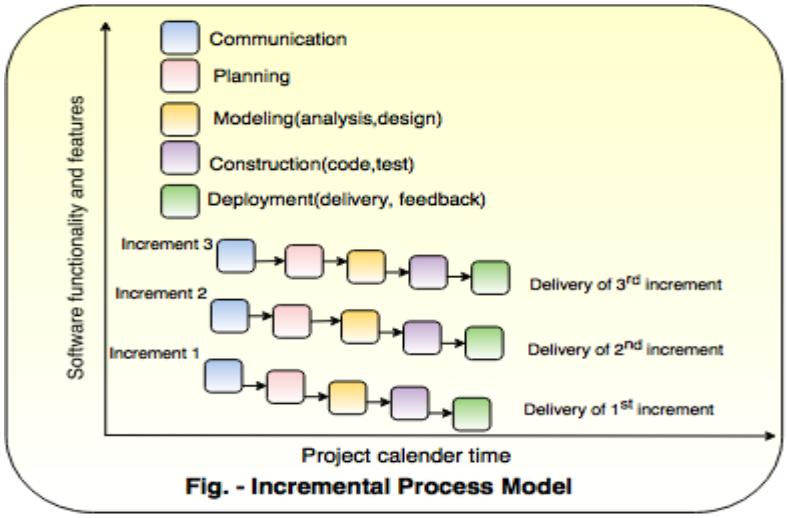


Fig. - Incremental Process Model

Following are the phases of Incremental model:

i) Communication
The software development starts with the communication between customer and developer.

ii) Planning
It consists of complete estimation, scheduling for project development.

iii) Modeling

- Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.
- The algorithm is a step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

iv) Construction

- Construction consists of code generation and the testing part.
- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not.
- Testing also checks that the program provides desired output.

v) Deployment

- Deployment step consists of delivering the product to the customer and taking feedback from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

Advantages of Incremental model

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

Disadvantages of the incremental model

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

6)RAD Model

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

The RAD model consist of following phases:

1) Business Modeling
2) Data modeling
3) Process modeling
4) Application generation
5) Testing and turnover

1) Business Modeling

- Business modeling consists of the flow of information between various functions in the project.

For example, what type of information is produced by every function and which are the functions to handle that information.

- It is necessary to perform complete business analysis to get the essential business information.

2) Data modeling

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

3) Process modeling

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

4) Application generation

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

5) Testing and turnover

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

Advantages of RAD Model

- The process of application development and delivery are fast.
- This model is flexible, if any changes are required.
- Reviews are taken from the clients at the staring of the development hence there are lesser chances to miss the requirements.

Disadvantages of RAD Model

- The feedback from the user is required at every development phase.
- This model is not a good choice for long term and large projects.

7)Iterative& Spiral Model

Iterative Model

- In Iterative model, the large application of software development is divided into smaller chunks and smaller parts of software which can be reviewed to recognize further requirements are implemented. This process is repeated to generate a new version of the software in each cycle of a model.
- With every iteration, development module goes through the phases i.e requirement, design, implementation and testing. These phases are repeated in iterative model in a sequence.

1) Requirement Phase
In this phase, the requirements for the software are assembled and analyzed. Generates a complete and final specification of requirements.

2) Design Phase
In this phase, a software solution meets the designed requirements which can be a new design or an extension of an earlier design.

3) Implementation and test phase
In this phase, coding for the software and test the code.

4) Evaluation
In this phase, software is evaluated, the current requirements are reviewed and the changes and additions in the requirements are suggested.

Advantages of an Iterative Model

- Produces working software rapidly and early in the software life cycle.
- This model is easy to test and debug in a smaller iteration.
- It is less costly to change scope and requirements.

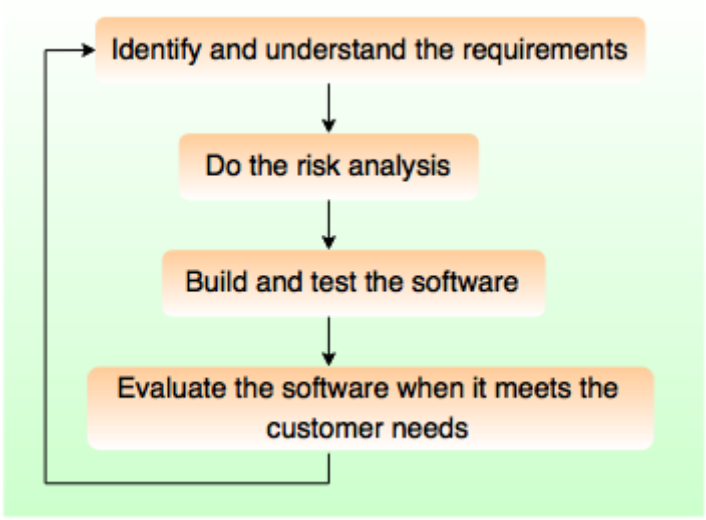
Disadvantages of an Iterative Model

- The system architecture is costly.
- This model is not suitable for smaller projects.

Spiral model

- It is a combination of prototype and sequential or waterfall model.
- This model was developed by Boehm.
- It is used for generating the software projects. This model is a risk driven process model.
- Every phase in the Spiral model is start with a design goal and ends with the client review.
- The development team in this model begins with a small set of requirements and for the set of requirements team goes through each development phase.
- The development team adds the functionality in every spiral till the application is ready.

Following are the steps involved in spiral model:



Phases of Spiral model are:

1) Planning
2) Risk Analysis
3) Engineering
4) Evaluation

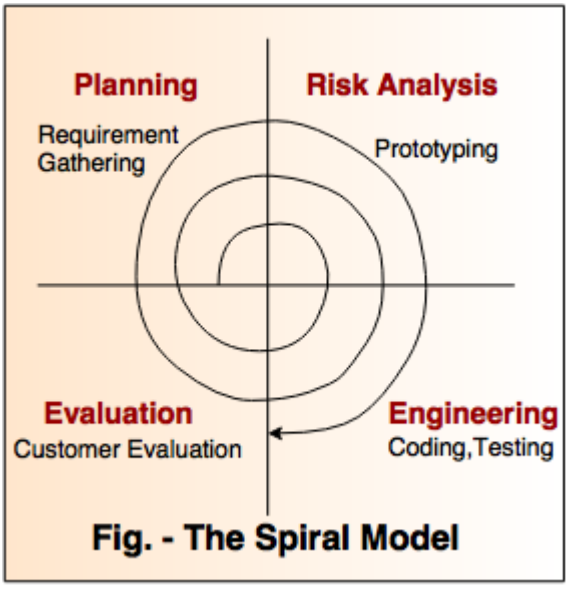


Fig. - The Spiral Model

1) Planning

- This phase, studies and collects the requirements for continuous communication between the customer and system analyst.
- It involves estimating the cost and resources for the iteration.

2) Risk Analysis
This phase, identifies the risk and provides the alternate solutions if the risk is found.

3) Engineering
In this phase, actual development i.e coding of the software is completed. Testing is completed at the end of the phase.

4) Evaluation
Get the software evaluated by the customers. They provide the feedback before the project continues to the next spiral.

Advantages of Spiral Model

- It reduces high amount of risk.
- It is good for large and critical projects.
- It gives strong approval and documentation control.
- In spiral model, the software is produced early in the life cycle process.

Disadvantages of Spiral Model

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

8)Prototype Model

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

The different phases of Prototyping model are:

1) Communication
2) Quick design
3) Modeling and quick design
4) Construction of prototype
5) Deployment, delivery, feedback

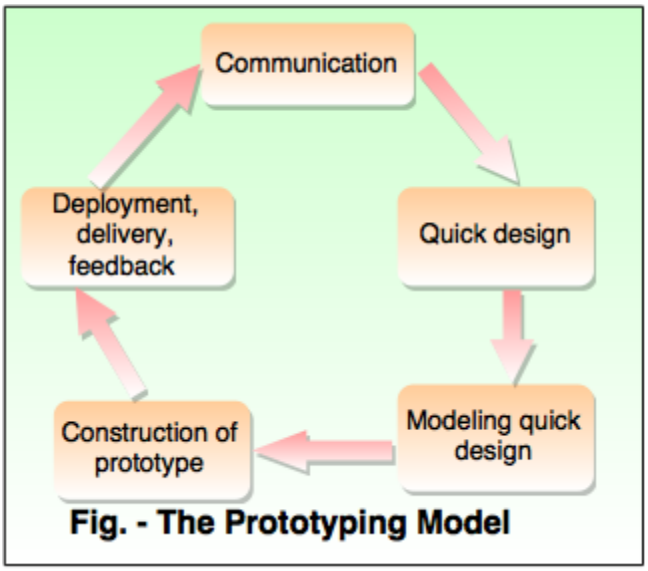


Fig. - The Prototyping Model

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

2. Quick design

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

3. Modeling quick design

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

4. Construction of prototype
The prototype is evaluated by the customer itself.

5. Deployment, delivery, feedback

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

Advantages of Prototyping Model

- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Earlier error detection takes place in this model.
- It gives quick user feedback for better solutions.
- It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

Disadvantages of Prototyping Model

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

REQUIREMENT ENGINEERING

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

Requirement Engineering Process

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

Feasibility study

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

Requirement Gathering

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

Software Requirement Specification

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

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

SRS should come up with following features:

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

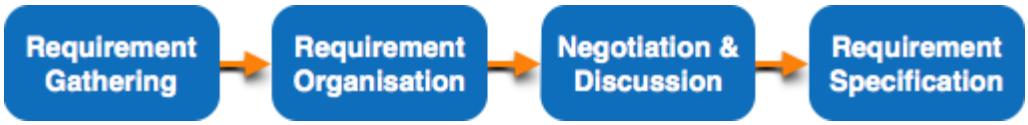
Software Requirement Validation

After requirement specifications are developed, the requirements mentioned in this document are validated. Requirements can be checked against following conditions -

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

Requirement Elicitation Process

Requirement elicitation process can be depicted using the folloiwng diagram:



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

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

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

Requirement Elicitation Techniques

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

There are various ways to discover requirements

Interviews

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

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

Surveys

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

Questionnaires

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

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

Task analysis

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

Domain Analysis

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

Brainstorming

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

Prototyping

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

Observation

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

THE ROLE OF REQUIREMENT ENGINEERING IN SOFTWARE DEVELOPMENT LIFE CYCLE

The Requirement Engineering (RE) is the most important phase of the Software Development Life Cycle (SDLC). This phase is used to translate the imprecise, incomplete needs and wishes of the potential users of software into complete, precise and formal specifications.

The specifications act as the contract between the software users and the developers. Therefore the importance of Requirement Engineering is enormous to develop effective software and in reducing software errors at the early stage of the development of software. Since Requirement Engineering (RE) has great role in different stages of the SDLC, its consideration in software development is crucial. There exist a number of approaches for requirement engineering.

TYPES OF SOFTWARE REQUIREMENTS

The most common types of software requirements are:

1. **Business Requirements (BR)**
    - These are high-level business goals of the organization building the product, or the customer who commissioned the project.
    - These are usually provided as a single page of high-level bullets.
2. **Market Requirements (MR)**

- These drill down into BRs, but still are high-level. In addition to business goals, they also outline market needs.
- These are usually provided as a prioritized bulleted list or table, and are usually less than 5 pages long.

3. **Functional Requirements (FR) – Use Cases**
   - These cover the functionality of the product in detail. Use cases are one of the best ways of documenting functional requirements.
   - Depending on the product being built, FRs can run several hundred pages.
4. **Non-Functional Requirements (NFR)**
   - These are not related to the "functionality" of the product – but cover goals such as Reliability, Scalability, Security, Integration, etc.
   - Many projects make the mistake of not specifying these explicitly.
5. **UI Requirements (UIR)**
   - User interface specs are not considered "requirements" in *traditional* requirements management theory.
   - Phooey! In my opinion, UI specs are indeed requirements (what else are they?) – and in fact should be considered an integral part of requirements for any software that has a UI.

STEPS INVOLVED IN REQUIREMENTS ENGINEERING

Requirement elicitation Developers and stakeholders meet, the latter are inquired concerning their needs and wants regarding the software product.

·**Requirements analysis and negotiation -** Requirements are identified (including new ones if the development is iterative) and conflicts with stakeholders are solved. Both written and graphical tools are successfully used as aids.

Examples of written analysis tools: use cases.

System modeling - Engineering fields (or specific situations) require the product to be ompletely designed and modeled before its construction or fabrication starts and, therefore, the design phase must be performed in advance. For instance, blueprints for a building must be elaborated before any contract can be approved and signed. Many fields might derive models of the system with the Life-cycle Modeling Language, whereas others, might use UML.

Requirements specification - Requirements are documented in a formal artifact called equirements Specification (RS). Nevertheless, it will become official only after validation. A RS can contain both written and graphical (models) information if necessary. Example: Software requirements specification (SRS).

Requirements validation - Checking that the documented requirements and models are consistent and meet the needs of the stakeholder. Only if the final draft passes the validation process, the RS becomes official.

UNIT-II

Analysis Modeling approach

- Analysis model operates as a link between the 'system description' and the 'design model'.
- In the analysis model, information, functions and the behavior of the system is defined and these are translated into the architecture, interface and component level design in the 'design modeling'.

Elements of the analysis model

1. Scenario based element

- This type of element represents the system user point of view.
- Scenario based elements are use case diagram, user stories.

2. Class based elements

- The object of this type of element manipulated by the system.
- It defines the object,attributes and relationship.
- The collaboration is occurring between the classes.
- Class based elements are the class diagram, collaboration diagram.

3. Behavioral elements

- Behavioral elements represent state of the system and how it is changed by the external events.
- The behavioral elements are sequenced diagram, state diagram.

4. Flow oriented elements

- An information flows through a computer-based system it gets transformed.
- It shows how the data objects are transformed while they flow between the various system functions.
- The flow elements are data flow diagram, control flow diagram.

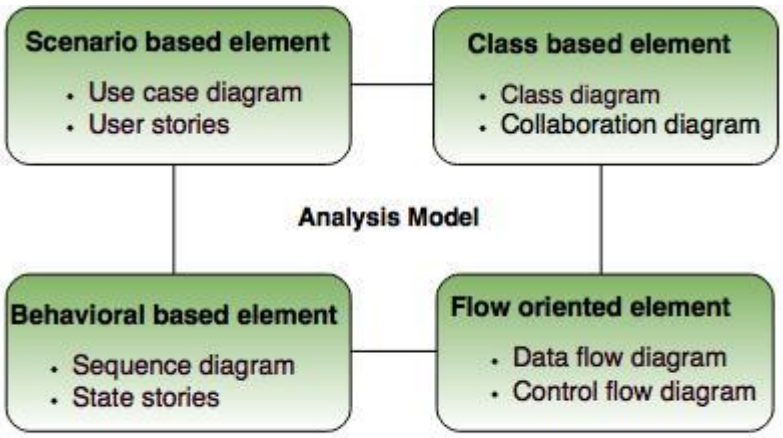


Fig. - Elements of analysis model

STRUCTURED ANALYSIS -OBJECT ORIENTED ANALYSIS
It does not support reusability of code. So, the time and cost of development is inherently high.

Structured Analysis vs. Object Oriented Analysis

The Structured Analysis/Structured Design (SASD) approach is the traditional approach of software development based upon the waterfall model. The phases of development of a system using SASD are −

- Feasibility Study
- Requirement Analysis and Specification
- System Design
- Implementation
- Post-implementation Review

Advantages/Disadvantages of Object Oriented Analysis

Advantages	Disadvantages
Focuses on data rather than the procedures as in Structured Analysis.	Functionality is restricted within objects. This may pose a problem for systems which are intrinsically procedural or computational in nature.
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the	It cannot identify which objects would generate an optimal system design.

system.

The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the system.

It allows effective management of software complexity by the virtue of modularity.

It can be upgraded from small to large systems at a greater ease than in systems following structured analysis.

The object-oriented models do not easily show the communications between the objects in the system.

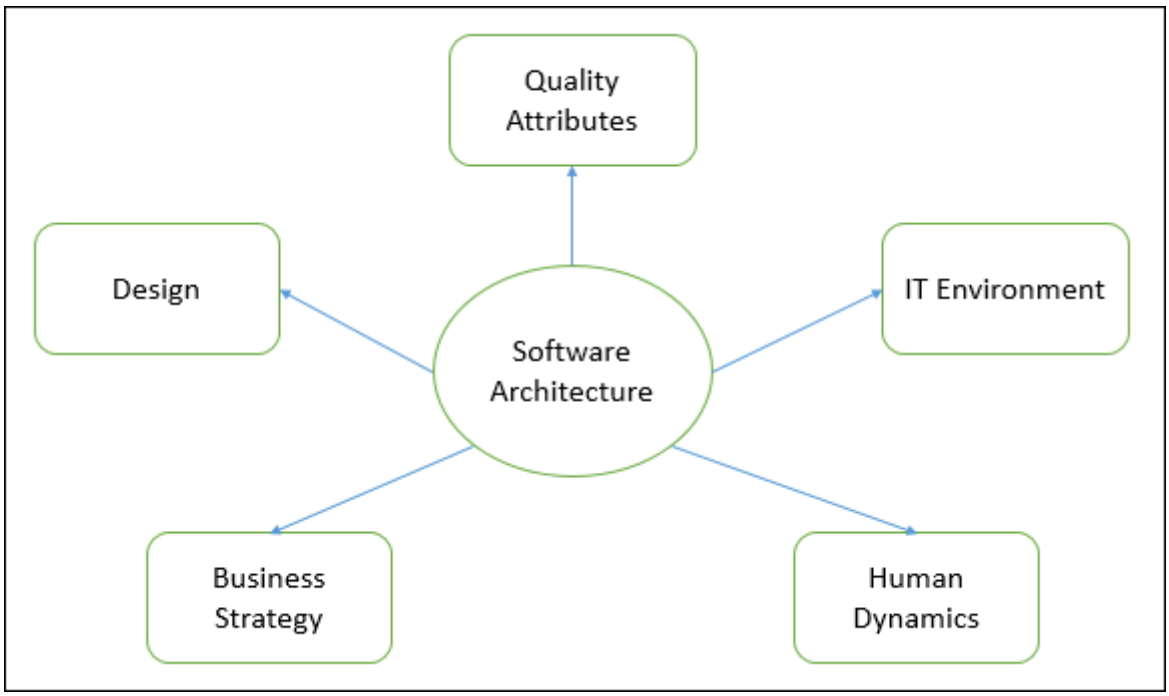
All the interfaces between the objects cannot be represented in a single diagram.

Advantages/Disadvantages of Structured Analysis

Advantages	Disadvantages
As it follows a top-down approach in contrast to bottom-up approach of object-oriented analysis, it can be more easily comprehended than OOA.	In traditional structured analysis models, one phase should be completed before the next phase. This poses a problem in design, particularly if errors crop up or requirements change.
It is based upon functionality. The overall purpose is identified and then functional decomposition is done for developing the software. The emphasis not only gives a better understanding of the system but also generates more complete systems.	The initial cost of constructing the system is high, since the whole system needs to be designed at once leaving very little option to add functionality later.
The specifications in it are written in simple English language, and hence can be more easily analyzed by non-technical personnel.	It does not support reusability of code. So, the time and cost of development is inherently high.

DESIGN AND ARCHITECTURAL ENGINEERING

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

Software Architecture

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
    - Selection of structural elements and their interfaces by which the system is composed.
    - Behavior as specified in collaborations among those elements.
    - Composition of these structural and behavioral elements into large subsystem.
    - Architectural decisions align with business objectives.
    - Architectural styles guide the organization.

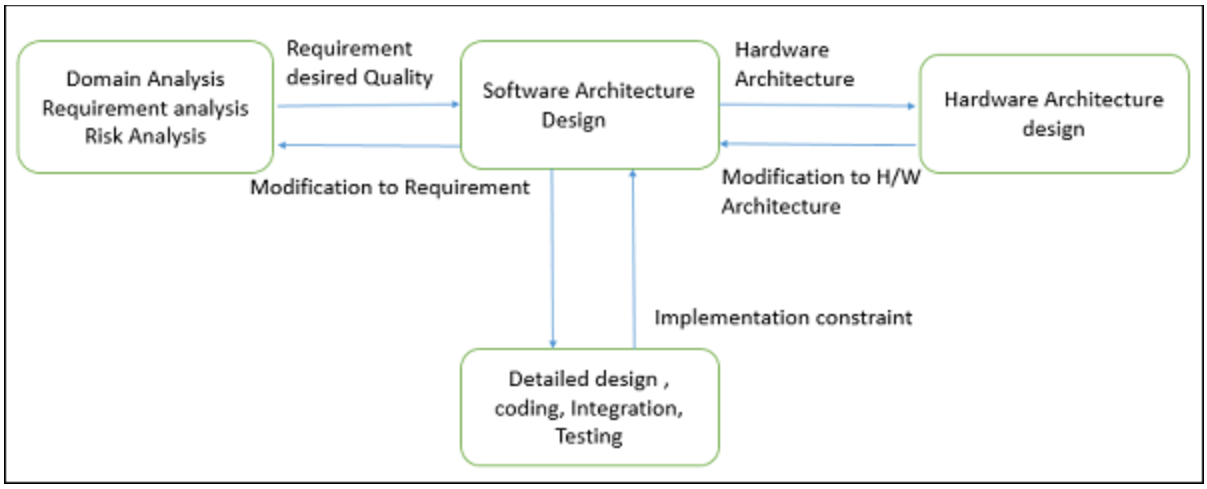
Software Design

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows −

- To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.
- Act as a blueprint during the development process.

- Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.



DESIGN PROCESS AND CONCEPTS

Introduction to design process

- The main aim of design engineering is to generate a model which shows firmness, delight and commodity.
- 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.

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 and the the program predictability.

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

- A 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 statement are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour 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.

BASIC ISSUES IN SOFTWARE DESIGN

A number of key issues must be dealt with when designing software. Some are quality concerns that all software must address—for example, performance, security, reliability, usability, etc. Another important issue is how to decompose, organize, and package software components.

2.1 Concurrency

Design for concurrency is concerned with decomposing software into processes, tasks, and threads and dealing with related issues of efficiency, atomicity, synchronization, and scheduling.

2.2 Control and Handling of Events

This design issue is concerned with how to organize data and control flow as well as how to handle reactive and temporal events through various mechanisms such as implicit invocation and call-backs.

2.3 Data Persistence

This design issue is concerned with how to handle long-lived data.

2.4 Distribution of Components

This design issue is concerned with how to distribute the software across the hardware (including computer hardware and network hardware), how the components communicate, and how middleware can be used to deal with heterogeneous software.

2.5 Error and Exception Handling and Fault Tolerance

This design issue is concerned with how to prevent, tolerate, and process errors and deal with exceptional conditions.

2.6 Interaction and Presentation

This design issue is concerned with how to structure and organize interactions with users as well as the presentation of information (

2.7 Security

Design for security is concerned with how to prevent unauthorized disclosure, creation, change, deletion, or denial of access to information and other resources. It is also concerned with how to tolerate security-related attacks or violations by limiting damage, continuing service, speeding repair and recovery, and failing and recovering securely. Access control is a fundamental concept of security, and one should also ensure the proper use of cryptology.

CHARACTERISTICS OF GOOD DESIGN

• **Correctness :-** A good design should correctly implement all the functionalities identified in the SRS document.

• **Understandability:**-A good design is easily understandable.

• **Efficiency :-**It should be efficient.

• **Maintainability:-**It should be easily amenable to change.

FUNCTION ORIENTED SYSTEM VS OBJECT SYSTEM:

1.FOD: The basic abstractions, which are given to the user, are real world functions.
OOD: The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.

2.FOD: Functions are grouped together by which a higher level function is Page on obtained.aneg of this technique isSA/SD.
OOD: Functions are grouped together on the basis of the data they operate since the classes are associated with their methods.

3.FOD: In this appproach the state information is often represented in a centralized shared memory.
OOD: In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.

4.FOD approach is mainly used for computation sensitive application,
OOD: whereas OOD approach is mainly used for evolving system which mimicks a business process or business case.

5. In FOD - we decompose in function/procedure level
OOD: - we decompose in class level

6. FOD: Top down Approach
OOD: Bottom up approach

7. FOD: It views system as Black Box that performs high level function and later decompose it detailed function so to be maaped to modules.
OOD: Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.

8. FOD: Begins by considering the use case diagrms and Scenarios.
OOD: Begins by identifiying objects and classes

Modularity, Cohesion, Coupling:

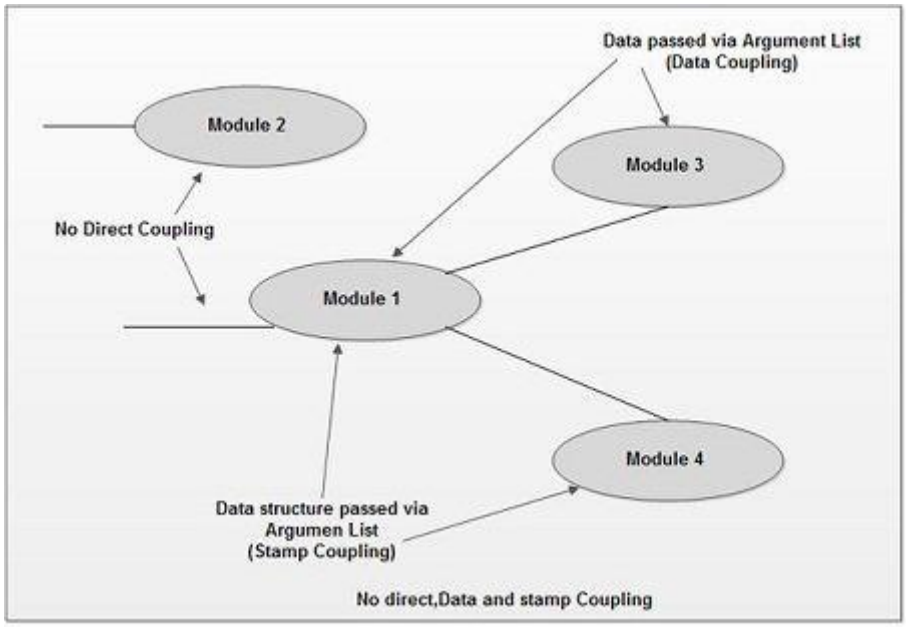
Modularization: Modularization is the process of dividing a software system into multiple independent modules where each module works independently. There are many advantages of Modularization in software engineering. Some of these are given below:

- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.

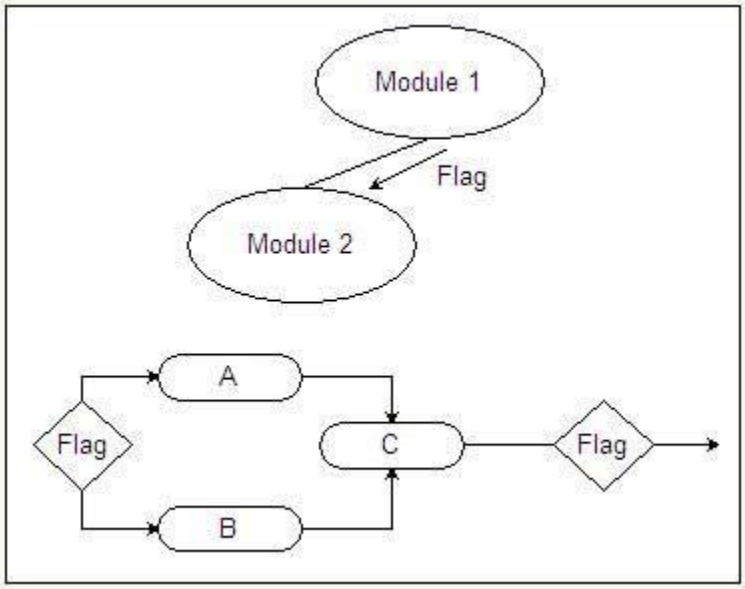
coupling

Coupling measures the degree of interdependence among the modules. Module coupling is categorized into the following types.

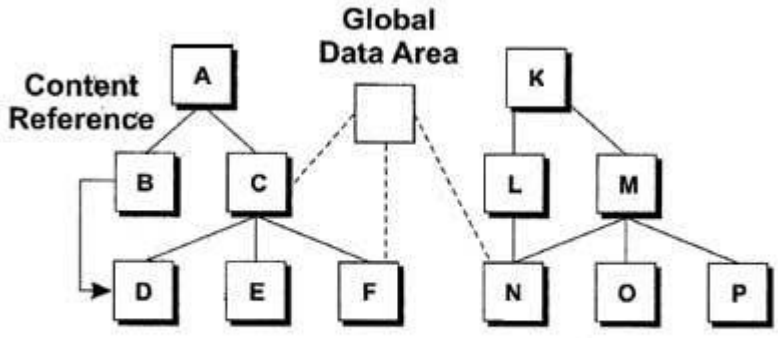
1. **No direct coupling:** Two modules are said to be 'no direct coupled'if they are independent of each other.



1. **Data coupling:** Two modules are said to be 'data coupled' if they use parameter list to pass data items for communication.
2. **Stamp coupling:** Two modules are said to be 'stamp coupled'if they communicate by passing a data structure that stores additional information than what is required to perform their functions.
3. **Control coupling:** Two modules are said to be 'control coupled'if they communicate (pass a piece of information intended to control the internal logic) using at least one 'control flag'. The control flag is a variable whose value is used by the dependent modules to make decisions.



1. **Contentcoupling:** Two modules are said to be 'content coupled' ifone module modifies data of some other module or one module is under the control of another module or one module branches into the middle of another module.



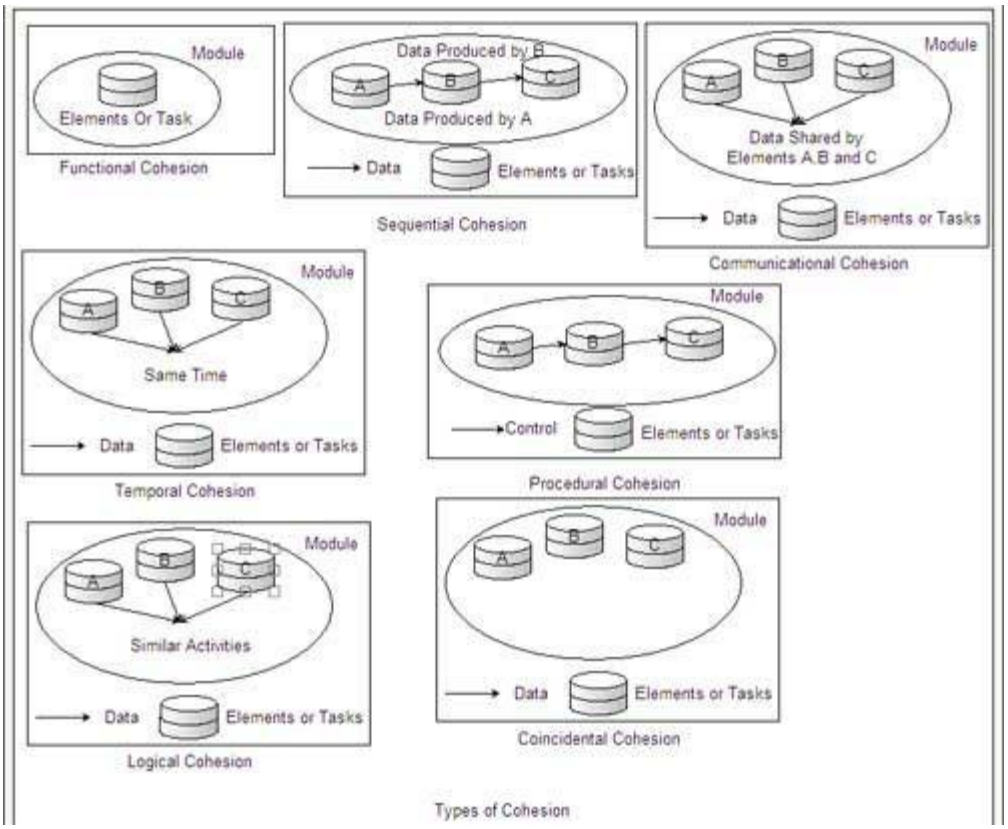
Content and Common Coupling

1. **Common coupling:** Two modules are said to be 'common coupled' if *they* both reference a common data block.

Cohesion

Cohesion measures the relative functional strength of a module. It represents the strength of bond between the internal elements of the modules. Various types of cohesion are listed below.

1. **Functional cohesion:** In this, the elements within the modules are concerned with the execution of a single function.
2. **Sequential cohesion:** In this, the elements within the modules are involved in activities in such a way that the output from one activity becomes the input for the next activity.
3. **Communicational cohesion:** In this, the elements within the modules perform different functions, yet each function references the same input or output information.
4. **Procedural cohesion:** In this, the elements within the modules are involved in different and possibly unrelated activities.
5. **Temporal cohesion:** In this, the elements within the modules contain unrelated activities that can be carried out at the same time.
6. **Logical cohesion:** In this, the elements within the modules perform similar activities, which are executed from outside the module.
7. **Coincidental cohesion:** In this, the elements within the modules perform activities with no meaningful relationship to one another.



- Layered technology

- Software engineering is a fully layered technology.
- To develop a software, we need to go from one layer to another.
- All these layers are related to each other and each layer demands the fulfillment of the previous layer.

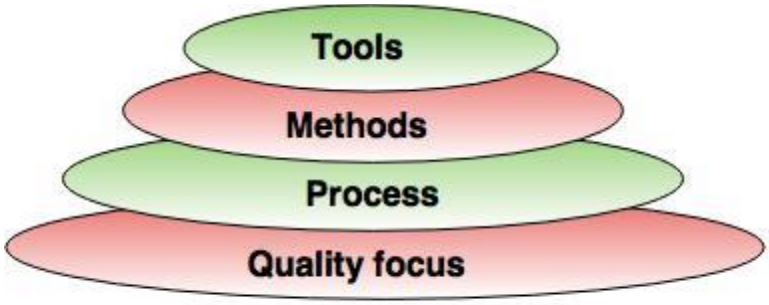


Fig. - Software Engineering Layers

The layered technology consists of:

1. Quality focus
The characteristics of good quality software are:

- Correctness of the functions required to be performed by the software.
- Maintainability of the software
- Integrity i.e. providing security so that the unauthorized user cannot access information or data.
- Usability i.e. the efforts required to use or operate the software.

2. Process

- It is the base layer or foundation layer for the software engineering.
- The software process is the key to keep all levels together.
- It defines a framework that includes different activities and tasks.
- In short, it covers all activities, actions and tasks required to be carried out for software development.

3. Methods

- The method provides the answers of all 'how-to' that are asked during the process.
- It provides the technical way to implement the software.
- It includes collection of tasks starting from communication, requirement analysis, analysis and design modelling, program construction, testing and support.

4. Tools

- The software engineering tool is an automated support for the software development.
- The tools are integrated i.e the information created by one tool can be used by the other tool.
- **For example:** The Microsoft publisher can be used as a web designing tool.

Real-time Software Design

Designing Realtime software involves several steps. The basic steps are listed below:

- Software Architecture Definition
- Co-Design
- Defining Software Subsystems
- Feature Design
- Task Design

Software Architecture Definition

This is the first stage of Realtime Software design. Here the software team understands the system that is being designed. The team also reviews at the proposed hardware architecture and develops a very basic software architecture. This architecture definition will be further refined in Co-Design.

Co-Design

Once the software architecture has been defined, the hardware and software teams should work together to associate software functionality to hardware modules.

Defining Software Subsystems

1. Determine all the features that the system needs to support.
2. Group the various features based on the type of work they perform. Identify various sub-systems by assigning one subsystem for one type of features. For example, for the Xenon switch the groups would be Call Handling, System Maintenance, Operator Interface etc.
3. Identify the tasks that will implement the software features. Clearly define the role of each task in its subsystem.

Feature Design

. Feature Design defines the software features in terms of message interactions between tasks. This involves detailed specification of message interfaces. The feature design is generally carried out in the following steps:

1. Specify the message interactions between different tasks in the system
2. Identify the tasks that would be controlling the feature. The controlling tasks would be keeping track of progress of feature. Generally this is achieved by running timers.
3. The message interfaces are defined in detail. All the fields and their possible values are identified.

Feature Design Guidelines

- Keep the design simple and provide a clear definition of the system composition.
- Do not involve too many tasks in a feature.
- Disintegrate big and complex features into small sub features.

Task Design

Designing a task requires that all the interfaces that the task needs to support should be very well defined. Make sure all the message parameters and timer values have been finalized.

Selecting the Task Type

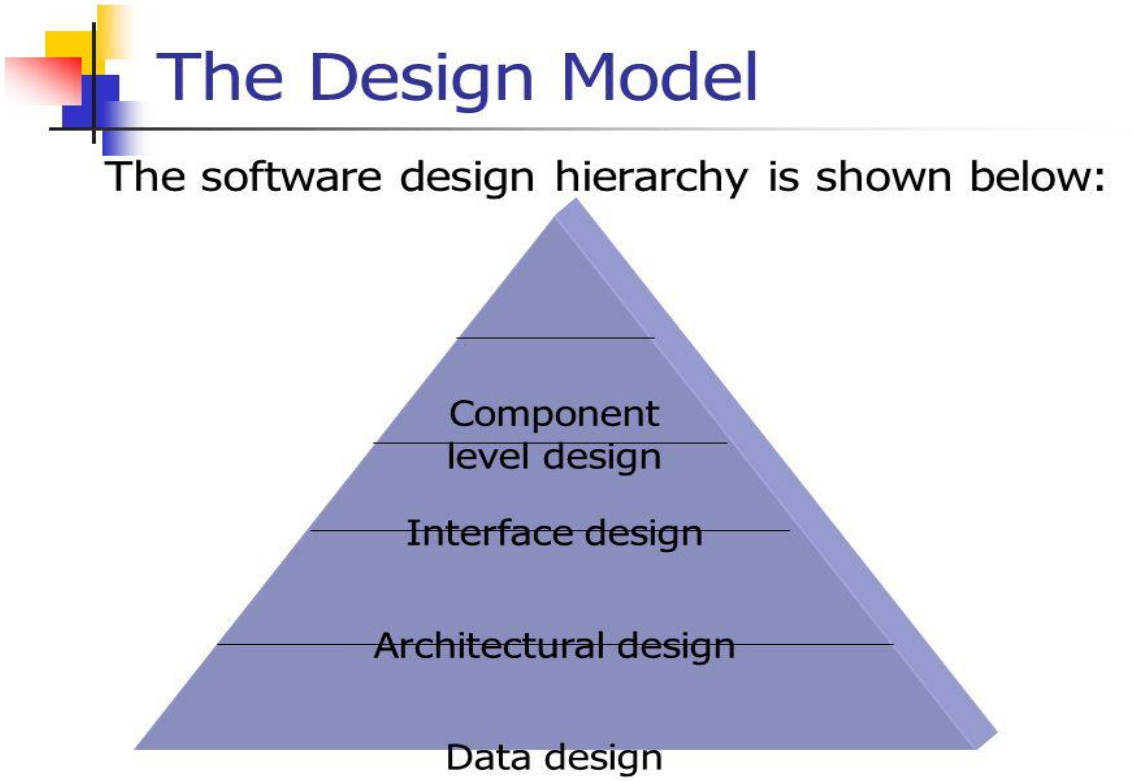
Once the external interfaces are frozen, select the type of task/tasks that would be most appropriate to handle the interfaces:

- **Single State Machine:** The tasks functionality can be implemented in a single state machine. **Multiple State Machines:** The task manages multiple state machines. Such tasks would typically include a dispatcher to distribute the received messages to an appropriate state machine**Multiple Tasks:**This type of tasks are similar to the multiple

state machine tasks discussed above. The main difference is that the task now manages multiple tasks.

- **Complex Task:** This type of task would be required in really complex scenarios. Here the manager task manages other tasks which might be managing multiple state machines.

Design Models:



THE COMPONENT-LEVEL DESIGN

Component design

- A software component is a modular building block for the computer software.
- Component is defined as a modular, deployable and replaceable part of the system which encloses the implementation and exposes a set of interfaces.

User Interface design

- User interface design helps in successing most of the software.
- It is part of the user and computer.
- Good interface design is user friendly.

Types of user interface:

1. Command Interpreter
Commands help the user to communicate with the computer system.

2. Graphical User Interfaces (GUI)

- It is another approach to communicate with system.
- It allows a mouse-based, window-menu-based systems as an interface.

Architecture design

Requirements of the software should be transformed into an architecture

this is accomplished through architectural design (also called **system design),** which acts as a preliminary 'blueprint' from which software can be developed.

architectural design as 'the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

An architectural design performs the following functions.

1. It defines an abstraction level at which the designers can specify the functional and performance behaviour of the system.

2. It acts as a guideline for enhancing the system (when ever required) by describing those features of the system that can be modified easily without affecting the system integrity.

3. It evaluates all top-level designs.

4. It develops and documents top-level design for the external and internal interfaces.

5. It develops preliminary versions of user documentation.

6. It defines and documents preliminary test requirements and the schedule for software integration.

7. The sources of architectural design are listed below.

8. Information regarding the application domain for the software to be developed

9. Using data-flow diagrams

10. Availability of architectural patterns and architectural styles.

Data Design:

Data design is the first design activity, which results in less complex, modular and efficient program structure. The information domain model developed during analysis phase is transformed into data structures needed for implementing the software.

1. The data structures needed for implementing the software as well-as the operations that can be applied on them should be identified.
2. A data dictionary should be developed to depict how different data objects interact with each other and what constraints are to be imposed on the elements of data structure.
3. Stepwise refinement should be used in data design process and detailed design decisions should be made later in the process.
4. Only those modules that need to access data stored in a data structure directly should be aware of the representation of the data structure.
5. A library containing the set of useful data structures along with the operations that can be performed on them should be maintained.
6. Language used for developing the system should support abstract data types.

DESIGN DOCUMENTATION

Design Documentation

Documentation is an essential part of any software development process that records the procedure of making the software. The design decisions need to be documented for any non–trivial software system for transmitting the design to others.

Usage Areas

Though a secondary product, a good documentation is indispensable, particularly in the following areas −

- In designing software that is being developed by a number of developers
- In iterative software development strategies
- In developing subsequent versions of a software project
- For evaluating a software
- For finding conditions and areas of testing
- For maintenance of the software.

Contents

A beneficial documentation should essentially include the following contents −

- **High–level system architecture** − Process diagrams and module diagrams
- **Key abstractions and mechanisms** − Class diagrams and object diagrams.
- **Scenarios that illustrate the behavior of the main aspects** − Behavioural diagrams

Features

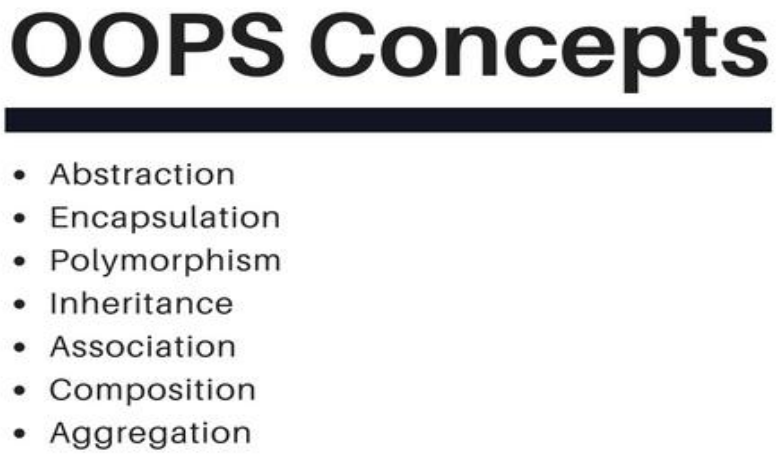
The features of a good documentation are −

- Concise and at the same time, unambiguous, consistent, and complete
- Traceable to the system's requirement specifications
- Well-structured
- Diagrammatic instead of descriptive

Unit -III

Object Oriented Concepts:

OOPS Concepts



Abstraction

Abstraction is the concept of hiding the internal details and describing things in simple terms. For example, a method that adds two integers. The internal processing of the method is hidden from the outer world.

Encapsulation

Encapsulation is the technique used to implement abstraction in object-oriented programming. Encapsulation is used for access restriction to class members and methods.

Access modifier keywords are used for encapsulation in object oriented programming. For example, encapsulation in java is achieved using private, protected and public keywords.

Polymorphism

Polymorphism is the concept where an object behaves differently in different situations. There are two types of polymorphism – compile time polymorphism and runtime polymorphism

Inheritance

Inheritance is the object oriented programming concept where an object is based on another object. Inheritance is the mechanism of code reuse. The object that is getting inherited is called superclass and the object that inherits the superclass is called subclass.

Association

Association is the OOPS concept to define the relationship between objects. Association defines the multiplicity between objects. For example Teacher and Student objects. There is one to many relationship between a teacher and students.

Aggregation

Aggregation is a special type of association. In aggregation, objects have their own life cycle but there is an ownership. Whenever we have "HAS-A" relationship between objects and ownership then it's a case of aggregation.

Fundatmental parts 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.

Elements of Object-Oriented System

Let us go through the characteristics of OO System −

- **Objects** − An object is something that is exists within problem domain and can be identified by data (attribute) or behavior. All tangible entities (student, patient) and some intangible entities (bank account) are modeled as object.
- **Attributes** − They describe information about the object.
- **Behavior** − It specifies what the object can do. It defines the operation performed on objects.
- **Class** − A class encapsulates the data and its behavior. Objects with similar meaning and purpose grouped together as class.
- **Methods** − Methods determine the behavior of a class. They are nothing more than an action that an object can perform.
- **Message** − A message is a function or procedure call from one object to another. They are information sent to objects to trigger methods. Essentially, a message is a function or procedure call from one object to another.

Structured Approach Vs. Object-Oriented Approach

Structured Approach	Object Oriented Approach
It works with Top-down approach.	It works with Bottom-up approach.
Program is divided into number of submodules or functions.	Program is organized by having number of classes and objects.
Function call is used.	Message passing is used.
Software reuse is not possible.	Reusability is possible.
Structured design programming usually left until end phases.	Object oriented design programming done concurrently with other phases.
Structured Design is more suitable for offshoring.	It is suitable for in-house development.
It shows clear transition from design to implementation.	Not so clear transition from design to implementation.
It is suitable for real time system, embedded system and projects where objects are not the most useful level of abstraction.	It is suitable for most business applications, game development projects, which are expected to customize or extended.
DFD & E-R diagram model the data.	Class diagram, sequence diagram, state chart

	diagram, and use cases all contribute.
In this, projects can be managed easily due to clearly identifiable phases.	In this approach, projects can be difficult to manage due to uncertain transitions between phase.

DataHiding and Class Hierarchy Creation

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside.
It permits the elements of the class to be accessed from outside only t
hrough the interface provided by the class.

Data Hiding

Typically, a class is designed such that its data (attributes) can be accessed only by its class methods and insulated from direct outside access. This process of insulating an object's data is calleddata hiding or information hiding.

Inheritance

Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super
-classes, and the new classes are called the derived classes/child classes/subclasses. The subclass can inherit or derive the attributes an
d methods of the super class(es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an "is a" relationship.

TypesofInheritance

♣**Single Inheritance**
: A subclass derives from a single super-class.
♣Multiple Inheritance
: A subclass derives from more than one super-classes.
♣Multilevel Inheritance
: A subclass derives from a super-class which in turn is derived from another class and so on.
♣Hierarchical Inheritance
: A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.
♣Hybrid Inheritance
: A combination of multiple and multilevel inheritance so as to form a lattice structure
.

Relationships:

A relationship is a connection between two or more UML model elements that adds semantic information to a model.

Association

Association is a relationship between two objects. In other words, association defines the multiplicity between objects. You may be aware of one-to-one, one-to-many, many-to-one, many-to-many all these words define an association between objects. Aggregation is a special form of association. Composition is a special form of aggregation.

Example: A Student and a Faculty are having an association.

Aggregation

Aggregation is a special case of association. A directional association between objects. When an object 'has-a' another object, then you have got an aggregation between them. Direction between them specified which object contains the other object. Aggregation is also called a "Has-a" relationship.

Composition

Composition is a special case of aggregation. In a more specific manner, a restricted aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.

Example: A class contains students. A student cannot exist without a class. There exists composition between class and students.

Abstraction

Abstraction is specifying the framework and hiding the implementation level information. Concreteness will be built on top of the abstraction. It gives you a blueprint to follow to while implementing the details. Abstraction reduces the complexity by hiding low level details.

Example: A wire frame model of a car.

Generalization

Generalization uses a "is-a" relationship from a specialization to the generalization class. Common structure and behaviour are used from the specializtion to the generalized class. At a very broader level you can understand this as inheritance. Why I take the term inheritance is, you can relate this term very well. Generalization is also called a "Is-a" relationship.

Example: Consider there exists a class named Person. A student is a person. A faculty is a person. Therefore here the relationship between student and person, similarly faculty and person is generalization.

Realization

Realization is a relationship between the blueprint class and the object containing its respective implementation level details. This object is said to realize the blueprint class. In other words, you can understand this as the relationship between the interface and the implementing class.

Example: A particular model of a car 'GTB Fiorano' that implements the blueprint of a car realizes the abstraction.

Dependency

Change in structure or behaviour of a class affects the other related class, then there is a dependency between those two classes. It need not be the same vice-versa. When one class contains the other class it this happens.

Example: Relationship between shape and circle is dependency.

Role of UML in OO Design

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

- UML stands for**Unified Modeling Language**.

- UML is different from the other common programming languages such as C++, Java, COBOL, etc.

- UML is a pictorial language used to make software blueprints.

- UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system.

Role of UML in OO Design

UML is a modeling language used to model software and non-software systems. Although UML is used for non-software systems, the emphasis is on modeling OO software applications. Most of the UML diagrams discussed so far are used to model different aspects such as static, dynamic, etc. Now whatever be the aspect, the artifacts are nothing but objects.

If we look into class diagram, object diagram, collaboration diagram, interaction diagrams all would basically be designed based on the objects.

Hence, the relation between OO design and UML is very important to understand. The OO design is transformed into UML diagrams according to the requirement. Before understanding the UML in detail, the OO concept should be learned properly. Once the OO analysis and design is done, the next step is very easy. The input from OO analysis and design is the input to UML diagrams.

Design Patterns

Patterns Design Pattern Template
Pattern name—describes the essence of the pattern in a short but expressive name

Intent—describes the pattern and what it does

Also-known-as—lists any synonyms for the pattern. Motivation—provides an example of the problem.Applicability—notes specific design situations in which the pattern is applicable.

Structure—describes the classes that are required to implement the pattern.

Participants—describes the responsibilities of the classes that are required to implement the pattern.

Collaborations—describes how the participants collaborate to carry out their responsibilities.

Consequences—describes the "design forces" that affect the pattern and the potential trade-offs that must be considered when the pattern is implemented.

Related patterns—cross-references related design patterns

Framework

- Framework is a set of cooperating classes that makes up a reusable design for a specific class of software."
- A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations.

- Differences between Frameworks and Design Patterns
  Frameworks are more concrete than Design Patterns
  - Can be embodied in code but only *examples* of patterns can be embodied in code.
  - A strength: can be written down in a programming language and not only studied, but executed and reused directly.
  - In contrast, design patterns have to be implemented (GoF patterns) each time they're used. (at a much higher level…)
  - Design patterns also explain the intent, tradeoff, and consequences of a design…
- Frameworks are larger architectural elements than design patterns.

  - Can be huge!
  - Industrial strength frameworks – if they are versatile and meet their objectives for reusability, … – can be very Very 'large.'
  - A typical framework contains several design patterns, but the reverse is never true
- Yet, frameworks are more specialized than design patterns.
  - Frameworks always have a particular application domain.
  - Frameworks have a particular focus.
  - Frameworks dictate an application architecture; design patterns do not.
    - Frameworks will consist of cooperating classes already set up to be used in a design via subclassing, etc.

Frameworks and Patterns

- Framework designers must address all these issues.
- Framework designer is much more likely to meet the goals of the framework (design and code reuse) by using design patterns.

Most frameworks usually use several design patterns.

OOAD - Object Oriented Analysis

In the system analysis or object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified.

The three analysis techniques that are used in conjunction with each other for object-oriented analysis are object modelling, dynamic modelling, and functional modelling.

Object Modelling

Object modelling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modelling can be visualized in the following steps −

- Identify objects and group into classes
- Identify the relationships among classes
- Create user object model diagram
- Define user object attributes
- Define the operations that should be performed on the classes
- Review glossary

Dynamic Modelling

After the static behavior of the system is analyzed, its behavior with respect to time and external changes needs to be examined. This is the purpose of dynamic modelling.

Dynamic Modelling can be defined as "a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world".

The process of dynamic modelling can be visualized in the following steps −

- Identify states of each object
- Identify events and analyze the applicability of actions
- Construct dynamic model diagram, comprising of state transition diagrams
- Express each state in terms of object attributes
- Validate the state–transition diagrams drawn

Functional Modelling

Functional Modelling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes as it moves between methods

The process of functional modelling can be visualized in the following steps −

- Identify all the inputs and outputs
- Construct data flow diagrams showing functional dependencies
- State the purpose of each function
- Identify constraints
- Specify optimization criteria

OOAD - Object Oriented Design

a detailed description is constructed specifying how the system is to be built on concrete technologies

The stages for object–oriented design can be identified as −

- Definition of the context of the system
- Designing system architecture
- Identification of the objects in the system
- Construction of design models
- Specification of object interfaces

System Design

Object-oriented system design involves defining the context of a system followed by designing the architecture of the system.

- **Context** − The context of a system has a static and a dynamic part. The static context of the system is designed using a simple block diagram of the whole system which is expanded into a hierarchy of subsystems. The subsystem model is represented by UML packages. The dynamic context describes how the system interacts with its environment. It is modelled using **use case diagrams**.
- **System Architecture** − The system architecture is designed on the basis of the context of the system in accordance with the principles of architectural design as well as domain knowledge. Typically, a system is partitioned into layers and each layer is decomposed to form the subsystems

Object Identification

The first step of object design is object identification. The objects identified in the object–oriented analysis phases are grouped into classes and refined so that they are suitable for actual implementation.

The functions of this stage are −

- Identifying and refining the classes in each subsystem or package
- Defining the links and associations between the classes
- Designing the hierarchical associations among the classes, i.e., the generalization/specialization and inheritances
- Designing aggregations

User Interface Design

User interface is the front-end application view to which user interacts in order to use the software. The software becomes more popular if its user interface is:

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

There are two types of User Interface:

1. **Command Line Interface:** Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.
2. **Graphical User Interface:** Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

User Interface Design Process:

The analysis and design process of a user interface is iterative and can be represented by a spiral model. The analysis and design process of user interface consists of four framework activities.

- **User, task, environmental analysis, and modeling:** Initially, the focus is based on the profile of users who will interact with the system, i.e. understanding, skill and knowledge, type of user, etc, based on the user's profile users are made into categories. From each category requirements are gathered. Based on the requirements developer understand how to develop the interface.
- Where will the interface be located physically?
- Will the user be sitting, standing, or performing other tasks unrelated to the interface?
- Does the interface hardware accommodate space, light, or noise constraints?
- Are there special human factors considerations driven by environmental factors?

2. **Interface Design:** The goal of this phase is to define the set of interface objects and actions i.e. Control mechanisms that enable the user to perform desired tasks
3. **Interface construction and implementation:** The implementation activity begins with the creation of prototype (model) that enables usage scenarios to be evaluated. As iterative design process continues a User Interface toolkit that allows the
4. **Interface Validation:** This phase focuses on testing the interface. The interface should be in such a way that it should be able to perform tasks correctly and it should be able to handle a variety of tasks.

User Interface Elements

When designing your interface, try to be consistent and predictable in your choice of interface elements..

Interface elements include but are not limited to:

- **Input Controls**: checkboxes, radio buttons, dropdown lists, list boxes, buttons, toggles, text fields, date field
- **Navigational Components**: breadcrumb, slider, search field, pagination, slider, tags, icons
- **Informational Components**: tooltips, icons, progress bar, notifications, message boxes, modal windows
- **Containers**: accordion

Input Controls

Element	Description
Checkboxes	Checkboxes allow the user to select one or more options from a set.
Radio buttons	Radio buttons are used to allow users to select one item at a time.

Element	Description
Dropdown lists	Dropdown lists allow users to select one item at a time,
List boxes	List boxes, like checkboxes, allow users to select a multiple items at a time.
Buttons	A button indicates an action upon touch and is typically labeled using text, an icon, or both.
Dropdown Button	The dropdown button consists of a button that when clicked displays a drop-down list of mutually exclusive items.

navigational Components

Element	Description
Search Field	A search box allows users to enter a keyword or phrase (query) and submit it to search the index with the intention of getting back the most relevant results.
Breadcrumb	Breadcrumbs allow users to identify their current location within the system by providing a clickable trail of proceeding pages to navigate by.
Pagination	Pagination divides content up between pages, and allows users to skip between pages or go in order through the content.
Tags	Tags allow users to find content in the same category. Some tagging systems also allow users to apply their own tags to content by entering them into the system.
Sliders	A slider, also known as a track bar, allows users to set or adjust a value

Information Components

Element	Description
Notifications	A notification is an update message that announces something new for the user to see. Notifications are typically used to indicate items such as, the successful completion of a task, or an error or warning message.
Progress Bars	A progress bar indicates where a user is as they advance through a series of steps in a process.
Message Boxes	A message box is a small window that provides information to users and requires them to take an action before they can move forward.

Designing the User Interface

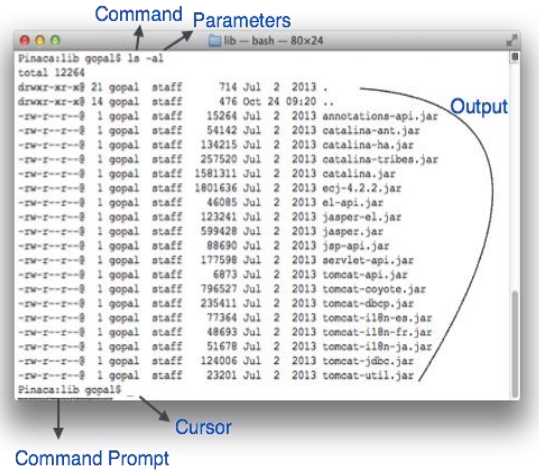
User interface is the front-end application view to which user interacts in order to use the software.

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:

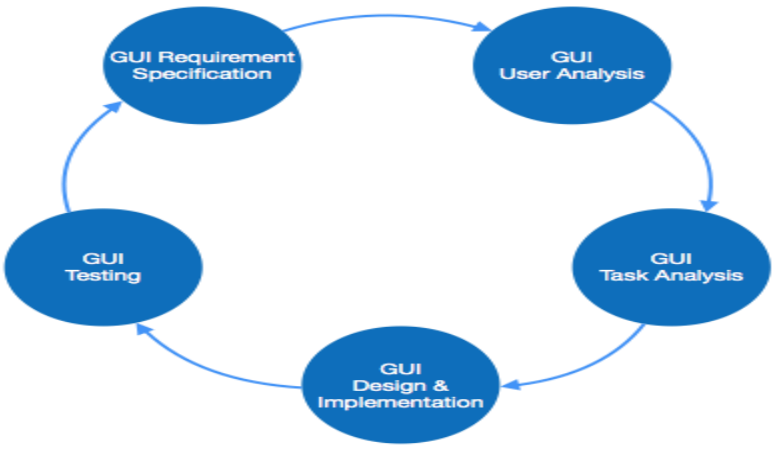


- Command Line Interface
- Graphical User Interface

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.



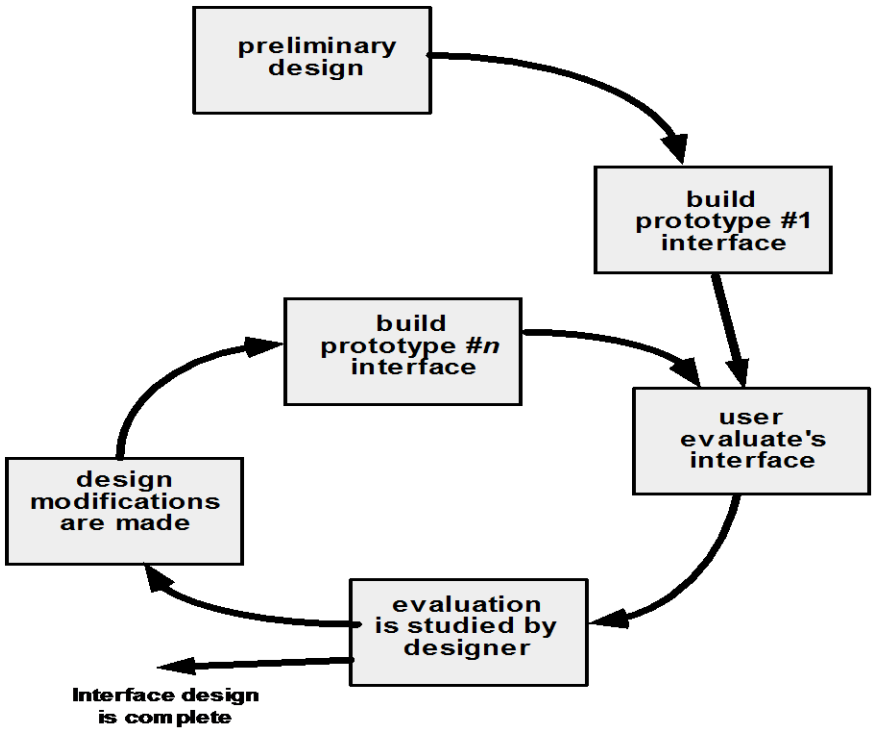
- **GUI Requirement Gathering** - The designers may like to have list of all functional and non-functional requirements of GUI. This can be taken from user and their existing software solution.
- **User Analysis** - The designer studies who is going to use the software GUI. The target audience matters 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.

USER INTERFACE DESIGN EVALUATION

Two interface design evaluation techniques are mentioned in this section, usability questionnaires and usability testing. The process of learning how to design good user interfaces often begins with learning to identify the weaknesses in existing products.



Once the first prototype is built, the designer can collect a variety of qualitative and quantitative data that will assess in evaluating the interface. Questions can be a simple Y/N response, numeric response, scaled response, Likert scale (strongly agree, etc.), percentage response, and open-ended ones.

GOLDEN RULES OF USER INTERFACE DESIGN

User Interface design

- User interface design helps in successing most of the software.
- It is part of the user and computer.
- Good interface design is user friendly.

Types of user interface:

1. Command Interpreter
Commands help the user to communicate with the computer system.

2. Graphical User Interfaces (GUI)

- It is another approach to communicate with system.
- It allows a mouse-based, window-menu-based systems as an interface.

The Golden Rules

The golden rules are known as interface design principles.

The golden rule are as follows:

1. Place the user in control

- The interaction should be defined in such a way that the user is not forced to implement unnecessary actions.
- The technical internal details must be hidden from the casual user.
- Design for the direct interaction with objects that appear on the screen.

2. Reduce the user's memory load

- The user interface must be designed in such a way that it reduces the demands on the user's short term memory.
- Create the meaningful defaults value as an advantage for the average users in the start of application.
- There must be a reset option for obtaining the default values.
- The shortcut should be easily remembered by the users.
- The interface screen should be friendly to users.

3. Make the interface consistent

- The system must allow the user to put task into meaningful context.
- Consistency should be maintained for all the interaction.
- Do not change the past system that is created by the user expectation unless there is a good reason to do that.

User Interface Models

There are three types of User Interfaces:

1. Command language
2. Menus
3. Graphical User Interface (GUI)

User interface models are used to analyzed and designed the user interface.

Following figure shows the different models of user interface:

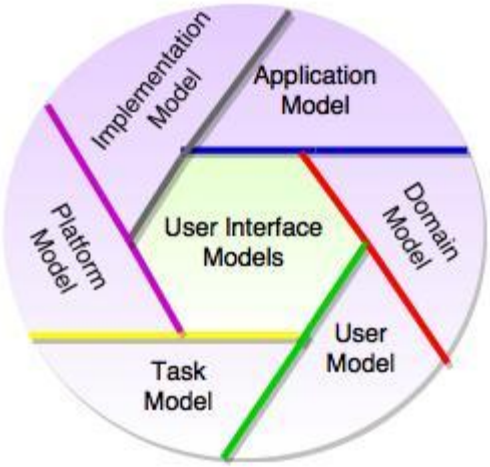


Fig. User Interface Models

1. Domain Model

- Domain model is implemented as an object model within a layer which describes aspects of knowledge or activity.
- This model is a representation of meaningful real-world concepts to the domain that need to be modeled in software.
- It is used to solve problems related to domain.

2. User Model

- User model represents the different features of end users and the roles they are playing within the organization.

3. Task Model

- Task model describes the tasks of an end user which performs and dictates what type of interaction capabilities must be designed.
- This model is the logical description of the activities and used for designing, analyzing and evaluating interactive software applications.

4. Platform Model

- Platform model is used to model the physical devices intended to host the application and defines how they interact with each other.

5. Implementation Model

- Implementation is a representation of how a system actually works and reflected in every interface element.
- This model is created by the software implementers who work on look and feel of the interface combined with all supporting information which describes the system syntax.

6. Application Model

- Application model performs a group of coordinated functions, tasks or activities for the benefit of the user.
- It refers to all applications involved with running the computer.

Usability

What is Usability?

- **Ease of learning**
- **Ease of use**
- **Ease of remembering**
- **Subjective satisfaction**
- **Efficiency of use**
- **Effectiveness of use**

Principle	Description
Layout	The interface should be a series of areas on the screen that are used consistently for different purposes—for example, a top area for commands and navigation, a middle area for information to be input or output, and a bottom area for status information.
Content awareness	Users should always be aware of where they are in the system and what information is being displayed.
Aesthetics	Interfaces should be functional and inviting to users through careful use of white space, colors, and fonts. There is often a tradeoff between including enough white space to make the interface look pleasing without losing so much space that important information does not fit on the screen.
User experience	Although ease of use and ease of learning often lead to similar design decisions, there is sometimes a tradeoff between the two. Novice users or infrequent users of software will prefer ease of learning, whereas frequent users will prefer ease of use.
Consistency	Consistency in interface design enables users to predict what will happen before they perform a function. It is one of the most important elements in ease of learning, ease of use, and aesthetics.
Minimal user effort	The interface should be simple to use. Most designers plan on having no more than three mouse clicks from the starting menu until users perform work.

UNIT IV

SOFTWARE CODING

This methodology refers to a set of well-documented procedures and guidelines used in the analysis, design, and implementation of programs.

1. In case the software development team is unable to understand user requirements correctly and further clarification is required, the queries are sent back to the user. In addition, the software development team also returns the requirements that are understood by them.
2. After the requirements are clearly understood by the software development team, the design and specifications are implemented in source code, supporting files, and the header files. Note that while writing the software code, the coding style guidelines should be followed. In some cases, there may be a proposal of change in hardware or software specifications. However, the requests for change are implemented only after the approval of the user.
3. When the software code is completely written, it is compiled along with other required files.
4. Code inspection and reviews are conducted after the compilation. These methods are used to correct and verify errors in the software code.
5. Software testing is carried out to detect and correct errors in each module of the software code.
6. After the software code is tested, the software is delivered to the user along with the relevant code files, header files, and documentation files.

Introduction to Software Measurement and Metrics

Software Measurement

Software measurements are of two categories, namely, direct measures and indirect measures. Direct measures include software processes like cost and effort applied and products like lines of code produced, execution speed, and other defects that have been reported. Indirect measures include products like functionality, quality, complexity, reliability, maintainability, and many more.

Measurement process is characterized by a set of five activities, which are listed below.

- **Formulation:** This performs measurement and develops appropriate metric for software under consideration.
- **Collection:** This collects data to derive the formulated metrics.
- **Analysis:** This calculates metrics and the use of mathematical tools.
- **Interpretation:** This analyzes the metrics to attain insight into the quality of representation.
- **Feedback:** This communicates recommendation derived from product metrics to the software team.

Software metrics

The goal of software metrics is to identify and control essential parameters that affect software development. Other objectives of using software metrics are listed below.

- Measuring the size of the software quantitatively.
- Assessing the level of complexity involved.
- Assessing the strength of the module by measuring coupling.
- Assessing the testing techniques.
- Specifying when to stop testing.
- Determining the date of release of the software.
- Estimating cost of resources and project schedule.

Software metrics help project managers to gain an insight into the efficiency of the software process, project, and product.

Software metrics is a standard of measure that contains many activities which involve some degree of measurement. It can be classified into three categories: product metrics, process metrics, and project metrics.

- **Product metrics** describe the characteristics of the product such as size, complexity, design features, performance, and quality level.
- **Process metrics** can be used to improve software development and maintenance. Examples include the effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process.
- **Project metrics** describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

What is Software Configuration Management?

Configuration Management helps organizations to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. It is abbreviated as the SCM process. It aims to control cost and work effort involved in making changes to the software system. The primary goal is to increase productivity with minimal mistakes.

Why do we need Configuration management?

The primary reasons for Implementing Software Configuration Management System are**:**

- There are multiple people working on software which is continually updating
- It may be a case where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently
- Changes in user requirement, policy, budget, schedule need to be accommodated.
- Software should able to run on various machines and Operating Systems
- Helps to develop coordination among stakeholders
- SCM process is also beneficial to control the costs involved in making changes to a system

Any change in the software configuration Items will affect the final product. Therefore, changes to configuration items need to be controlled and managed.

Tasks in SCM process

Configuration Identification

Baselines

Change Control

Configuration Status Accounting

Configuration Audits and Reviews

Configuration Identification:

Configuration identification is a method of determining the scope of the software system. With the help of this step, you can manage or control something even if you don't know what it is. It is a description that contains the CSCI type (Computer Software Configuration Item), a project identifier and version information.

Activities during this process:

- Identification of configuration Items like source code modules, test case, and requirements specification.
- Identification of each CSCI in the SCM repository, by using an object-oriented approach
- The process starts with basic objects which are grouped into aggregate objects. Details of what, why, when and by whom changes in the test are made
- Every object has its own features that identify its name that is explicit to all other objects
- List of resources required such as the document, the file, tools, etc.

Example:

Instead of naming a File login.phpits should be named login_v1.2.php where v1.2 stands for the version number of the file

Instead of naming folder "Code" it should be named "Code_D" where D represents code should be backed up daily.

Baseline:

A baseline is a formally accepted version of a software configuration item. It is designated and fixed at a specific time while conducting the SCM process. It can only be changed through formal change control procedures.

Activities during this process:

- Facilitate construction of various versions of an application
- Defining and determining mechanisms for managing various versions of these work products
- The functional baseline corresponds to the reviewed system requirements
- Widely used baselines include functional, developmental, and product baselines

In simple words, baseline means ready for release.

Change Control:

Change control is a procedural method which ensures quality and consistency when changes are made in the configuration object. In this step, the change request is submitted to software configuration manager.

Activities during this process:

- Control ad-hoc change to build stable software development environment. Changes are committed to the repository
- The request will be checked based on the technical merit, possible side effects and overall impact on other configuration objects.
- It manages changes and making configuration items available during the software lifecycle

Configuration Status Accounting:

Configuration status accounting tracks each release during the SCM process. This stage involves tracking what each version has and the changes that lead to this version.

Activities during this process:

- Keeps a record of all the changes made to the previous baseline to reach a new baseline
- Identify all items to define the software configuration
- Monitor status of change requests
- Complete listing of all changes since the last baseline
- Allows tracking of progress to next baseline
- Allows to check previous releases/versions to be extracted for testing

Configuration Audits and Reviews:

Software Configuration audits verify that all the software product satisfies the baseline needs. It ensures that what is built is what is delivered.

Activities during this process:

- Configuration auditing is conducted by auditors by checking that defined processes are being followed and ensuring that the SCM goals are satisfied.
- To verify compliance with configuration control standards. auditing and reporting the changes made
- SCM audits also ensure that traceability is maintained during the process.
- Ensures that changes made to a baseline comply with the configuration status reports
- Validation of completeness and consistency

Participant of SCM process:

Following are the key participants in SCM

1. Configuration Manager

- Configuration Manager is the head who is Responsible for identifying configuration items.
- CM ensures team follows the SCM process
- He/She needs to approve or reject change requests

2. Developer

- The developer needs to change the code as per standard development activities or change requests. He is responsible for maintaining configuration of code.
- The developer should check the changes and resolves conflicts

3. Auditor

- The auditor is responsible for SCM audits and reviews.
- Need to ensure the consistency and completeness of release.

4. Project Manager:

- Ensure that the product is developed within a certain time frame
- Monitors the progress of development and recognizes issues in the SCM process
- Generate reports about the status of the software system
- Make sure that processes and policies are followed for creating, changing, and testing

5. User

The end user should understand the key SCM terms to ensure he has the latest version of the software

SOFTWARE PROJECT MANAGEMENT:

Need of software project management

Software is said to be an intangible product. Software development is a kind of all new stream in world business and there's very little experience in building software products. Most software products are tailor made to fit client's requirements. The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one.

Software Project Manager

A software project manager is a person who undertakes the responsibility of executing the software project.

Software Management Activities

Software project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management. Project management activities may include:

- **Project Planning**
- **Scope Management**
- **Project Estimation**

Project Planning

Software project planning is task, which is performed before the production of software actually starts.

Scope Management

It defines the scope of project; this includes all the activities, process need to be done in order to make a deliverable software product. During Project Scope management, it is necessary to -

- Define the scope
- Decide its verification and control

- Divide the project into various smaller parts for ease of management.
- Verify the scope
- Control the scope by incorporating changes to the scope

Project Estimation

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation**

  Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software.

- **Effort estimation**

  The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software

  Time estimation

  Once size and efforts are estimated, the time required to produce the software can be estimated

  The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

- **Cost estimation**

  This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

  - Size of software
  - Software quality
  - Hardware
  - Additional software or tools, licenses etc.
  - Skilled personnel with task-specific skills
  - Travel involved
  - Communication
  - Training and support

Introduction to Software Testing

ntroduction:-

Testing is a process of executing a program with the aim of finding error. To make our software perform well it should be error free.If testing is done successfully it will remove all the errors from the software.

Principles of Testing:-

(i) All the test should meet the customer requirements
(ii) To make our software testing should be performed by third party
(iii) Exhaustive testing is not possible.As we need the optimal amount of testing based on the risk assessment of the application.
(iv) All the test to be conducted should be planned before implementing it
(v) It follows pareto rule(80/20 rule) which states that 80% of errors comes from 20% of program components.
(vi) Start testing with small parts and extend it to large parts.

software Validation

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC

- Validation ensures the product under development is as per the user requirements.
- Validation answers the question – "Are we developing the product which attempts all that user needs from this software ?".
- Validation emphasizes on user requirements.

Software Verification

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verification answers the question– "Are we developing this product by firmly following all design specifications ?"
- Verifications concentrates on the design and system specifications.

Black-box testing

It is carried out to test functionality of the program. It is also called ‘Behavioral’ testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested ‘ok’, and problematic otherwise.

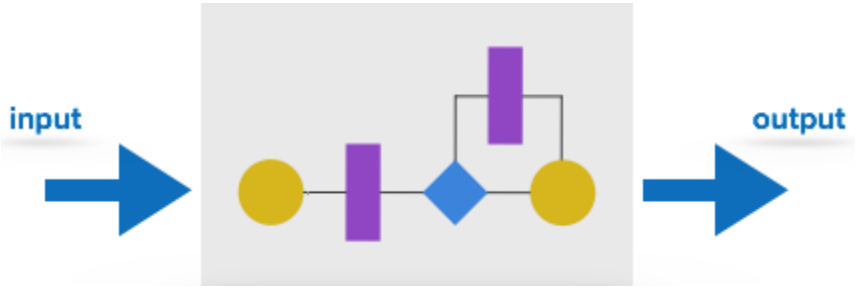
In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

Black-box testing techniques:

- **Equivalence class** - The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.
- **Boundary values** - The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.
- **Cause-effect graphing** - In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.
- **Pair-wise Testing** - The behavior of software depends on multiple parameters. In pairwise testing, the multiple parameters are tested pair-wise for their different values.
- **State-based testing** - The system changes state on provision of input. These systems are tested based on their states and input.

White-box testing

It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as ‘Structural’ testing.



In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

The below are some White-box testing techniques:

- **Control-flow testing** - The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.
- **Data-flow testing** - This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

Unit Testing

While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

Integration Testing

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updation etc.

System Testing

The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

- **Functionality testing** - Tests all functionalities of the software against the requirement.
- **Performance testing** - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. Performance testing is done by means of load testing and stress testing where the software is put under high user and data load under various environment conditions.
- **Security & Portability** - These tests are done when the software is meant to work on various platforms and accessed by number of persons.

Acceptance Testing

When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

- **Alpha testing** - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.
- **Beta testing** - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.

Regression Testing

Whenever a software product is updated with new code, feature or functionality, it is tested thoroughly to detect if there is any negative impact of the added code. This is known as regression testing.

SOFTWARE MAINTENANCE

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

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

Types of maintenance

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

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

- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

  Training facility is provided if required, in addition to the hard copy of user manual.

- **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

https://www.tutorialspoint.com/software_engineering/software_project_management.htm

SOFTWARE ENGINEERING

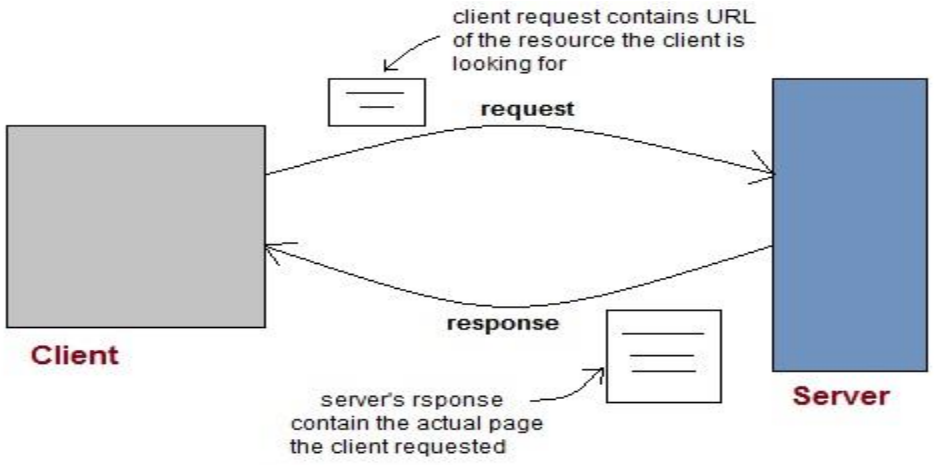
UNIT V

WEB ENGINEERING

Web Engineering is the application of systematic and quantifiable approaches(concepts, methods ,techniques, tools)to cost-(p,,q,)effective requirements analysis, design, implementation, testing, operation, and maintenance of high-quality Web applications.

INTRODUCTION TO WEB

Web consists of billions of clients and server connected through wires and wireless networks. The web clients make requests to web server. The web server receives the request, finds the resources and return the response to the client. When a server answers a request, it usually sends some type of content to the client. The client uses web browser to send request to the server. The server often sends response to the browser with a set of instructions written in HTML(HyperText Markup Language). All browsers know how to display HTML page to the client.



Web Application

A website is a collection of static files(webpages) such as HTML pages, images, graphics etc. A **Web application** is a web site with dynamic functionality on the server. **Google**, **Facebook**, **Twitter** are examples of web applications

HTTP (Hypertext Transfer Protocol)

- HTTP is a protocol that clients and servers use on the web to communicate.
- It is similar to other internet protocols such as SMTP(Simple Mail Transfer Protocol) and FTP(File Transfer Protocol) but there is one fundamental difference.
- HTTP is a **stateless protocol** i.e HTTP supports only one request per connection. This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
- The client sends an HTTP request and the server answers with an HTML page to the client, using HTTP.

CHARACTERISTICS OF WEB APPLICATIONS

Characteristics of web applications :

- Product related characteristics
- Use related characteristics
- Development related characteristics
- Evolution related characteristics

Product Related Characteristics

It is the integral part of web application. It consists of: Present, Hypertext, and Content.

- **Present:**. The application must be attractive, impressive and according to fashion trend going on in market.
- **Hypertext:** Hypertext is the base of web application. The basic elements of hypertext are: link, node and anchor
- **Content:** Content is the informational part. Content generation, integration and updating and availability is an important factor. It contains document, table, text, graphics, and multimedia. arranged.

Use Related Characteristics

It is difficult to predict the usage frequency of a web application because is varies according to the user, devices used by the users etc..

- **Natural Content:** It includes the geographical location from where the web applications are accessed and availability of the web application.
- **Social Content:** It is related to user specific aspect. There are thousands of competitive web applications around the globe, the user need spontaneous and immediate benefits.
- **Technical Content:** It is related to network of web application and the devices where web application are used. Connection bandwidth, stability, reliability etc..

Development Related Characteristics

It includes: Development team, Development process, Technical infrastructure and Integration.

- **Development Team:** Development team must be highly knowledgeable in their field. There must be proficient designers, database developers, IT experts, hypertext experts, application developers.
- **Development Process:** The development process must be flexible. There must be parallel processes of development.
- **Technical Infrastructure:** The web application must be bugs free and development should be under time limit. Server and Browser are the two external components that should be considered at the time of development.
- **Integration:** The web application must have support for integration with already existing system or with external content and services.

Evolution Related Characteristics

- As per the changes in requirements there occurs some changes or upgradations in the web application. This evolution may concern all the other three characteristics viz. Product, Use and Development. Market competition or short time development may cause the changes.

CATEGORIES OF WEB APPLICATIONS :

We can categorize web applications as follows:

- Document centric web application
- Interactive web application
- Transactional web application
- Work-flow based web application
- Collaborative web application
- Portal-oriented web application
- Ubiquitous web application
- Knowledge-based web application

Document centric web application

Document centric web sites are static html documents stored on web server that is sent directly to the client on request. The web pages are manually updated with the help of respective tools. These applications are static, simple, stable and take less time to respond

Interactive web application

Interactive web applications are offered by CGI, HTML Forms. It includes radio buttons, selection menus, forms etc. These applications are simple and fast. In this kind of application the web pages and links are generated according to user input.

Transactional web application

These kind of web applications have facility of modification by user. These applications are more interactive and support structured queries from database. The database system handle data consistently and efficiently.

Work-flow based web application

These kind of web applications are capable of handing the workflow among companies, private authorities or public authorities. Web services are included for interoperability. B2B e-commerce solutions are best example of such applications.

Collaborative web application

These kind of applications are mainly used as group applications where group communications are important part. Chat rooms, online forums, e-learning websites or websites where information are shared with option of editing like Wikipedia

Portal oriented web application

This kind of web applications are those where single access point is there to separate different sources of information and services. Search engines, community portals etc. are best examples of portal oriented application.

Ubiquitous web application

These kind of applications provides customized facilities for any device from anywhere at any time. It has limited interaction facility and support limited device. It require advance knowledge of context where the web application is being used for dynamic adjustment. Services based on location is an example.

Knowledge-based web application

This kind of application is used for providing knowledge for both human and machine. The knowledge management is based on semantic web technologies. Mining the web, linking and reusing knowledge are a few examples.

WORKING OF WEB APPLICATION :

Web applications are usually coded in browser-supported language such as JavaScript and HTML as these languages rely on the browser to render the program executable. Some of the applications are dynamic, requiring server-side processing. Others are completely static with no processing required at the server.

The web application requires a web server to manage requests from the client, an application server to perform the tasks requested, and, sometimes, a database to store the information. Application server technology ranges from ASP.NET, ASP and ColdFusion, to PHP and JSP.

Here's what a typical web application flow looks like:

1. **User** triggers a request to the **web server** over the **Internet**, either through a web browser or the application's user interface
2. **Web server** forwards this request to the appropriate **web application server**
3. **Web application server** performs the requested task – such as querying the **database** or processing the data – then generates the results of the requested data
4. **Web application server** sends results to the **web server** with the requested information or processed data
5. **Web server** responds back to the client with the requested information that then appears on the user's display

Example of a web application

Web applications include online forms, shopping carts, word processors, spreadsheets, video and photo editing, file conversion, file scanning, and email programs such as Gmail, Yahoo and AOL. Popular applications include Google Apps and Microsoft 365.

Advantages of Web Apps:

- **A Better User Experience** – With responsive design, it's a lot easier and cheaper to make a web based system user friendly across multiple platforms and various screen sizes.
- **Flexible Access** – Employees can work from anywhere with internet access.
- **Client Secure Login** – Impress clients with a modern web portal and improve customer service with automated processes.
- **Easy Setup** – It takes a couple of minutes to setup a new user; provide a URL, username and password and they're away.
- **Always Up To Date** – As everyone is accessing the same version of the web app via a URL, they will always be accessing the most up-to-date version of the software.
- **Storage Increase** – With the availability of the cloud, storage space is virtually infinite.

Disadvantages Of Web Apps:

- **Internet reliance** – Whilst 4G & Wi-Fi internet access is available in many locations, if you happen to lose connection you will not be able to access your web app.
- **Security** – Whilst many business people may believe that data is less secure in a cloud environment, There are ways in which you can reduce risk of a data breach, such as SSL enforcement for a secure HTTPS access to your app.
- **Reduced Speed** – It's likely that a web app will operate at a slightly slower speed than one hosted on a server locally.
- **Browser Support** – Unfortunately, we don't all use the same browser. This means during development you'll need to ensure your app is supported across a variety of browsers.

SOFTWARE ENGINEERING TRENDS

1. **Soft Trends**,,

The broad characteristics of the new systems we build (and test),

- e.g., •Emergent requirements•
- Autonomy of action
    - The anthropological and sociological characteristics of the new generation of people who do software engineering work

2.Hard Trends

Technical aspects of next generation software intensive systems "The technical directions that software engineering process, methods, and tools will take

Soft Trends - I

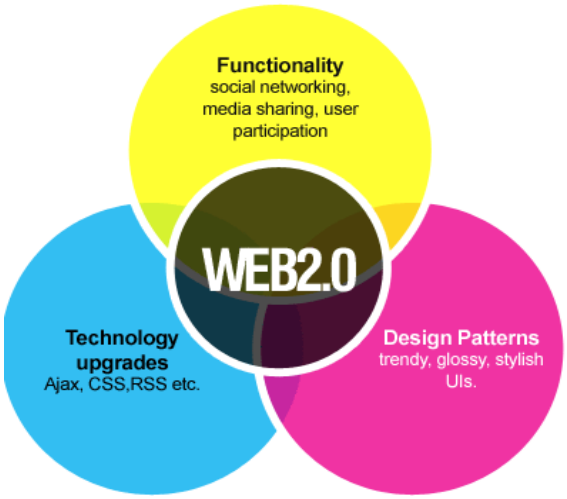
- Connectivity and collaboration(enabled by high bandwidth communication),,
- software teams that do not occupy the same physical space (telecommuting and part-time employment in a local context).
- Globalization leads to a diverse workforce "in terms of language, culture, problem resolution, management philosophy, communication priorities, and person-to-person interaction,,
- An aging population implies that many experienced software engineers and managers will be leaving the field over the coming decade „We need viable mechanisms that capture the knowledge of these aging managers and technologists
- Consumer spending in emerging economies will double to well over $9 trillion.„Some of this spending will be applied to products and services that have a digital component that is software-based

Soft Trends - II

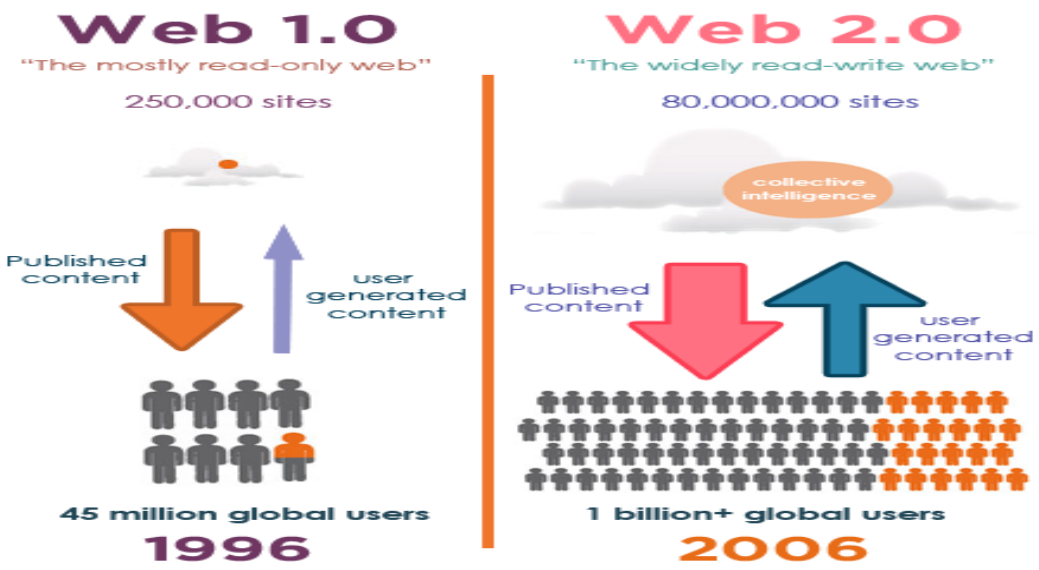
- People and Teams,,
- As systems grow in size, teams grow in number, geographical distribution, and culture,,
- As systems grow in complexity, team interfaces become pivotal to success.
- As systems become pervasive, teams must manage emergent requirements.
- As systems become more open, what is a team?

WHAT IS A WEB 2.0 TECHNOLOGY?

It's important to note that Web 2.0 frameworks only deal with the design and use of websites, without placing technical demands on designers.



it's a simply improved version of the first world wide web, characterized specifically by the change from static to dynamic or user-generated content and also the growth of social media. The concept behind Web 2.0 refers to rich web applications, web- oriented architecture and social web. It refer to changes in the ways web pages are designed and used by the users, without any change in any technical specifications.



Web 2.0 examples include hosted services (Google Maps),Web applications (Google Docs, Flickr), Video sharing sites (You Tube), wikis (MediaWiki), blogs (WordPress), social networking (Facebook), folksonomies (Delicious), Microblogging (Twitter), podcasting (Podcast Alley) & content hosting services and many more.

OPEN SOURCE SOFTWARE DEVELOPMENT

Free OSS

Software that gives users rights to run, copy, distribute, change and improve it as they see it, without them asking permission from or make payments to any external group or person

- Freedom to study the code
- Freedom to improve the program
- Freedom to run the program anytime, for any purpose on any machine.
- Freedom to redistribute.

Open Source Software

- Open source software is FreeOSS that uses any license approved by the Open Source Initiative (OSI) from their list of approved open source licenses
- www.opensource.org/licenses/

Free OS Software

- ❖ Apache
- ❖ BIND
- ❖ Emacs

- FreeBSD
- Ghostscript
- Jakarta
- Jboss
- LaTex
- Linux
- MySQL
- Open Office
- Perl
- Samba
- Sendmail
- Snort

SECURITY ENGINEERING

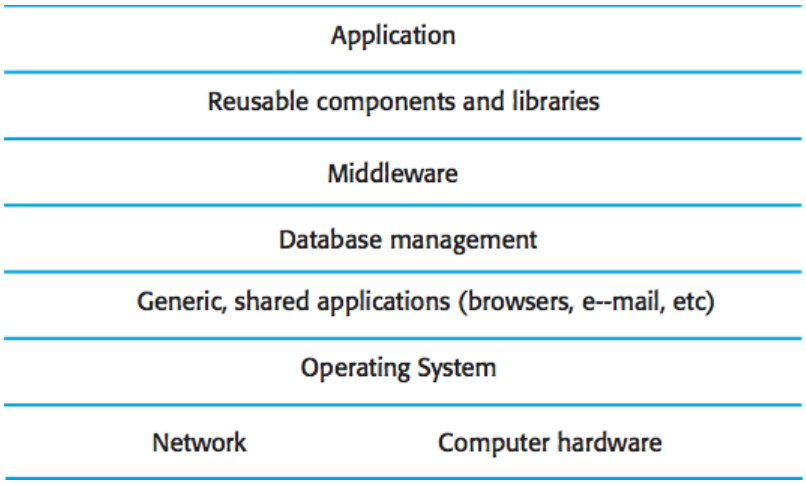
Security engineering is a sub-field of the broader field of computer security. It encompasses **tools, techniques and methods** to support the development and maintenance of systems that can **resist malicious attacks** that are intended to damage a computer-based system or its data.

Dimensions of security:

- **Confidentiality** Information in a system may be disclosed or made accessible to people or programs that are not authorized to have access to that information.
- **Integrity** Information in a system may be damaged or corrupted making it unusual or unreliable.
- **Availability** Access to a system or its data that is normally available may not be possible.

Three levels of security:

- **Infrastructure security** is concerned with maintaining the security of all systems and networks that provide an infrastructure and a set of shared services to the organization.
- **Application security** is concerned with the security of individual application systems or related groups of systems.
- **Operational security** is concerned with the secure operation and use of the organization's systems.



Application security

It is a software engineering problem where the system is designed to resist attacks. **Infrastructure security** is a systems management problem where the infrastructure is configured to resist attacks.

System security management

It involves **user and permission management** (adding and removing users from the system and setting up appropriate permissions for users), **software deployment and maintenance** (installing application software and middleware and configuring these systems so that vulnerabilities are avoided), **attack monitoring, detection and recovery** (monitoring the system for unauthorized access, design strategies for resisting attacks and develop backup and recovery strategies).

Operational security

It is primarily a human and social issue, which is concerned with ensuring the people do not take actions that may compromise system security. Users sometimes take insecure actions to make it easier for them to do their jobs. There is therefore a trade-off between system security and system effectiveness.

SERVICE ORIENTED SOFTWARE ENGINEERING -

- Service-oriented software engineering incorporates the best of these two paradigms. Initially, SOSE was based on services computing, but it evolved to include cloud computing.
- In SOSE, a service-oriented architecture (SOA) provides the architectural style, standard protocols, and interfaces required for application development, and cloud computing delivers the needed services to users through virtualization and resource pooling.

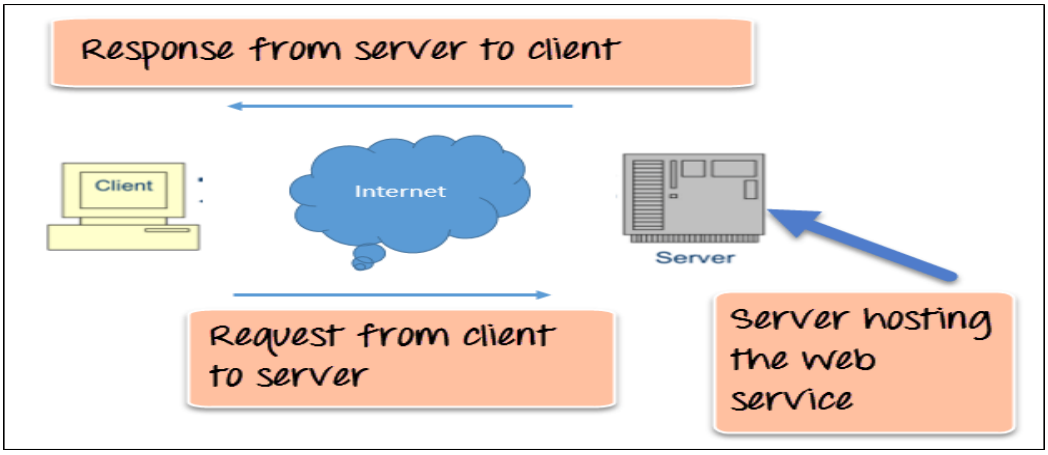
- Combining services and cloud computing in a software engineering framework can help application developers and service providers meet the individual challenges of each paradigm.

WHAT IS WEB SERVICE?

Web service is a standardized medium to propagate communication between the client and server applications on the World Wide Web.

A web service is a software module which is designed to perform a certain set of tasks.

- The web services can be searched for over the network and can also be invoked accordingly.
- When invoked the web service would be able to provide functionality to the client which invokes that web service.



- Web services use something known as SOAP (Simple Object Access Protocol) for sending the XML data between applications.
- The data is sent over normal HTTP. The data which is sent from the web service to the application is called a SOAP message.
- The SOAP message is nothing but an XML document. Since the document is written in XML, the client application calling the web service can be written in any programming language.

TYPE OF WEB SERVICE

There are mainly two types of web services.

1. SOAP web services.
2. Restful web services.

SOAP (Simple Object Access Protocol)

- SOAP is known as a transport-independent messaging protocol. SOAP is based on transferring XML data as SOAP Messages.
- Each message has something which is known as an XML document. Only the structure of the XML document follows a specific pattern, but not the content.
- The best part of Web services and SOAP is that its all sent via HTTP, which is the standard web protocol.

WEB SERVICE ARCHITECTURE

1. **Provider** - The provider creates the web service and makes it available to client application who want to use it.
2. **Requestor** - A requestor is nothing but the client application that needs to contact a web service. The client application can be a .Net, Java, or any other language based application which looks for some sort of functionality via a web service.
3. **Broker** - The broker is nothing but the application which provides access to the UDDI. The UDDI, as discussed in the earlier topic enables the client application to locate the web service.

SOFTWARE AS A SERVICE

- Software that performs various tasks is not on the client machine. Instead, third-party service providers host and manage the software services in the cloud.
- SaaS includes both software components (for application developers) and applications (for users). An SaaS application is often a service-oriented program so that it is easy to integrate with other SaaS applications.

SERVICE ORIENTED ARCHITECTURE

- Service developers follow SOA, an architectural model for creating and sharing computing processes, packaged as services .
- Each service is an independent software entity with a well-defined standard interface that provides certain functions over networks.
- Developers can dynamically compose services as a workflow, which forms the basis of an application.
- In this context, software itself can be a service—a self-contained, stateless, and platform-independent entity with a URL, an interface, and functions that can be described and discovered as XML data

SOA-based application development is through service discovery and composition, which involves three stakeholders:

- A ***service provider*** (or developer) is the party who develops and hosts the service.
- A ***service consumer*** is a person or program that uses a service to build an application.

- A ***service broker*** helps service providers publish and market their services and helps service consumers discover and use the available services.

services have several attractive characteristics. They are

- ***loosely coupled***—there are no direct dependencies among individual services;
- ***abstract***—beyond the SLA description, a service hides its logic from the outside world;
- ***reusable***—services aim to support potential reuse;
- ***composable***—a service can comprise other services, and developers can coordinate and assemble services to form a composite;
- ***stateless***—to remain loosely coupled, services do not maintain state information specific to an activity, such as a service request; and
- ***discoverable***—services let a service consumer use mechanisms to discover and understand their descriptions.

CLOUD COMPUTING

- Cloud computing enables convenient, on-demand network access to a shared pool of configurable computing resources, such as networks, servers, storage, applications, and services, which the cloud system can rapidly provision and release automatically.
- Cloud computing lets a consumer (user or program) request computing capabilities as needed, across networks anytime
    - .There are four cloud types. A ***public cloud*** provides services to the public. A ***private cloud*** provides services to only users within one organization. A ***community cloud*** provides services to a specific community of organizations and individuals. A ***hybrid cloud*** is any combination of the first three types.

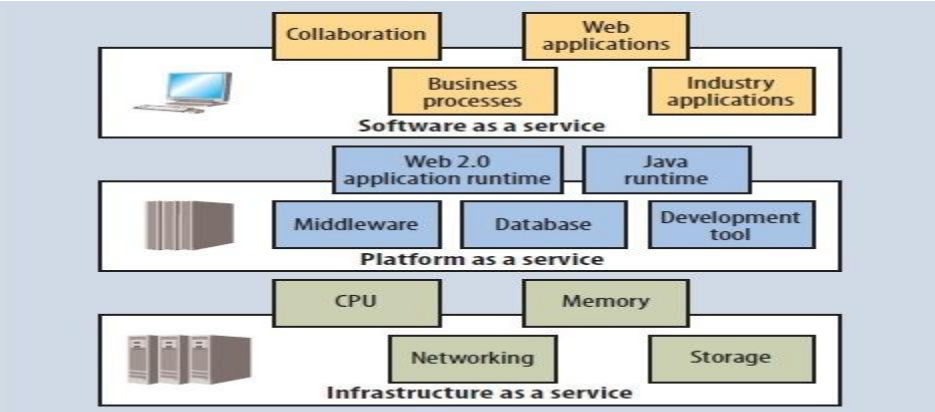


Figure 1. Cloud computing architecture. The top layer allows users and application developers to access services that third parties host and manage. The second layer consists of computing platforms and development services, and the third layer provides the computing resources that users can tap on demand.

ASPECT-ORIENTEDSOFTWAREDEVELOPMENT

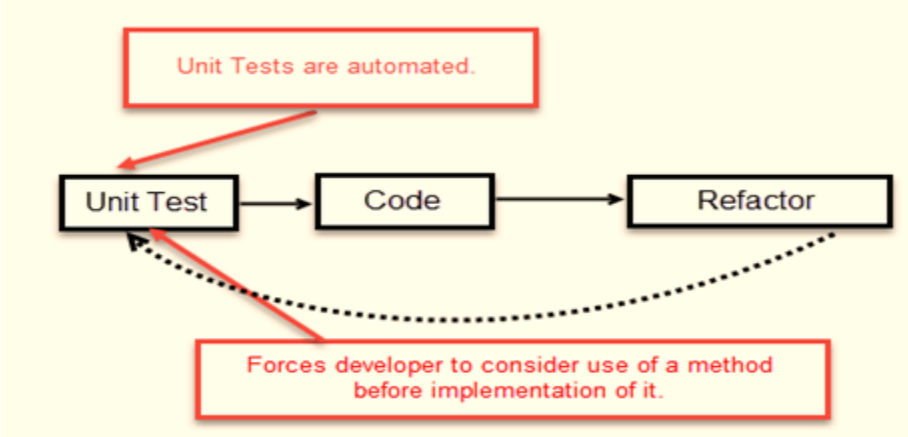
- Approach to software development that addresses modularization of software systems′Separates out or isolates supporting functions from the program's main business logic′Concerns are addressed independently′The dominant programming paradigm is object-oriented software development (OOSD)
- AOSD is not meant to replace OOSD but instead to build upon and improve
- AOSD supports separation of concerns that OOSD handles poorly

RESOURCES

AOSD.net contains much information on the topic, it is also a great starting point because it links to separate resources′AspectJ is an AOP extension for Java′Since its release in 2001, it quickly became the de facto standard for AOP′Integrated into the Eclipse IDE Displays the cross-cutting structure′Many programming languages have implemented AOP either within the language or as a separate library.

TEST DRIVEN DEVELOPMENT

- TDD can be defined as a programming practice that instructs developers to write new code only if an automated test has failed. This avoids duplication of code. TDD means "Test Driven Development". The primary goal of TDD is to make the code clearer, simple and bug-free.
- Test-Driven Development starts with designing and developing tests for every small functionality of an application. In TDD approach, first, the test is developed which specifies and validates what the code will do.
- In the normal Software Testing process, we first generate the code and then test. Tests might fail since tests are developed even before the development. In order to pass the test, the development team has to develop and refactors the code. Refactoring a code means changing some code without affecting its behavior



- The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. (Tests are nothing but requirement conditions that we need to test to fulfill them).
- Test-Driven development is a process of developing and running automated test before actual development of the application. Hence, TDD sometimes also called as **Test First Development.**

SOCIAL COMPUTING

- Social computing refers to an area of computer science that is the intersection of social behavior and computational systems1. So, social computing implies two components: a social behavior component and a computational system or technical component. The technical component provides the environ-ment in which people interact
  .
- Another definition of social computing is, "Social computing is the use of technology in networked communication systems by communities of people for one or more goals."2Social computing takes many forms including social networks, RSS, blogs, search engines, podcasts, wikis, and social bookmarking (or tagging).

social computing include social network analysis, and social identity theory. The foundation of social computing lies in the underlying social network.

The following are some of the **characteristics of social network analysis:**

- Actors and their actions are viewed as interdependent rather than independent, autonomous unit
- Relational ties (linkages) between actors are channels for transfer or "flow" of resources (either material or nonmaterial)
- Network models focusing on individuals view the network structural environment as providing opportunities for or constraints on individual action
- Network models conceptualize structure (social, economic, political, and so forth) as lasting patterns of relations among actors