



Your roots to success....

# **NARSIMHA REDDY ENGINEERING COLLEGE**

## **UGC AUTONOMOUS INSTITUTION**

Maisammaguda (V), Kompally - 500100, Secunderabad, Telangana State, India

UGC - Autonomous Institute

Accredited by NBA & NAAC with 'A' Grade

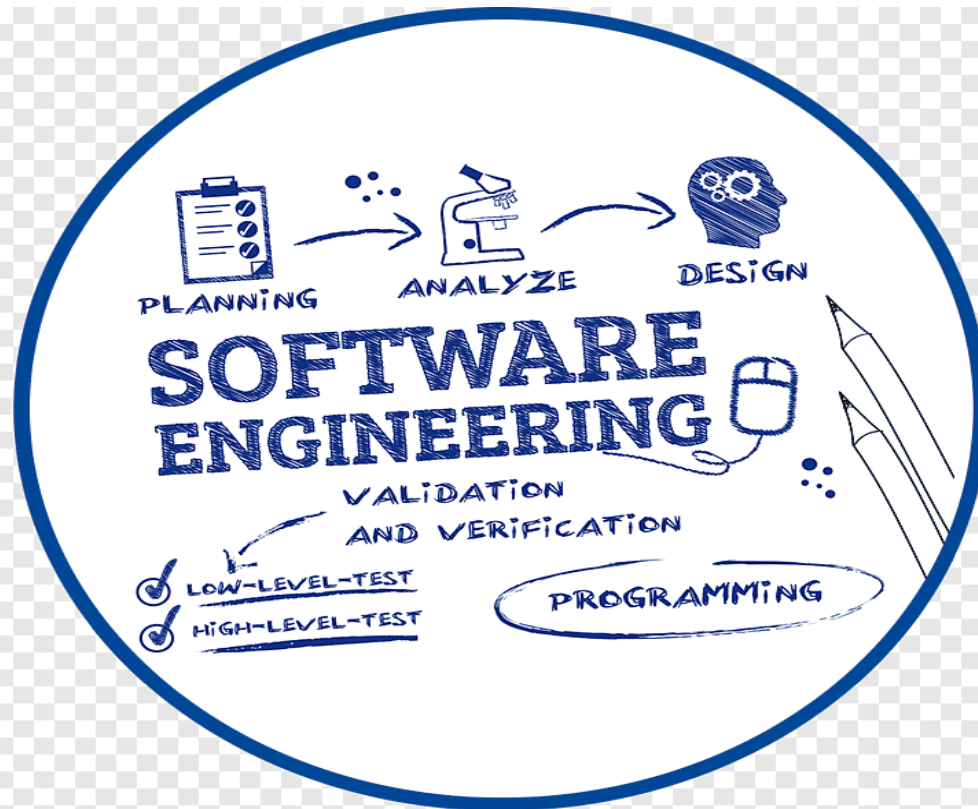
Approved by AICTE

Permanently affiliated to JNTUH

## **INFORMATION TECHNOLOGY**

### **SOFTWARE ENGINEERING (23IT501)**

#### **III-YEAR I SEM**



SC  
CHAPT.

# CONTENTS

Software Engineering Definition

Nature of Software Engineering

Software Myths

Layered Technology

Process Framework

CMMI (Capability Maturity Model Integration)

Process Patterns, Assessments

Personal and team Process models

Waterfall Model, Incremental Process Model, Evolutionary Process Model

Unified Process



# 1. SOFTWARE ENGINEERING

Software engineering is defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements.

It is an engineering stream dedicated to software development. Software programs can be developed without S/E principles and methodologies but they are indispensable if we want to achieve good quality software in a cost effective manner

There are various definitions for software engineering

# IEEE DEFINITION

IEEE, in its standard 610.12-1990, defines software engineering as the application of a systematic, disciplined, which is a computable approach for the development, operation, and maintenance of software.



# IEEE

# FRITZ BAUER DEFINITION

The establishment and used standard engineering principles. It helps you to obtain, economically, software which is reliable and works efficiently on the real machines.




# BOEHM DEFINITION

The practical application of scientific knowledge to the creative design and building of computer programs. It also includes associated documentation needed for developing, operating, and maintaining them.



## 2. CHANGING NATURE OF SOFTWARE ENGINEERING

Now-a-days, the software landscape has been completely changed . There are 7 various software each for a specific use

- System Software
  - Application Software
  - Engineering and Scientific Software
  - Embedded Software
  - Product Line Software
  - Web Application
  - AI Software
- 

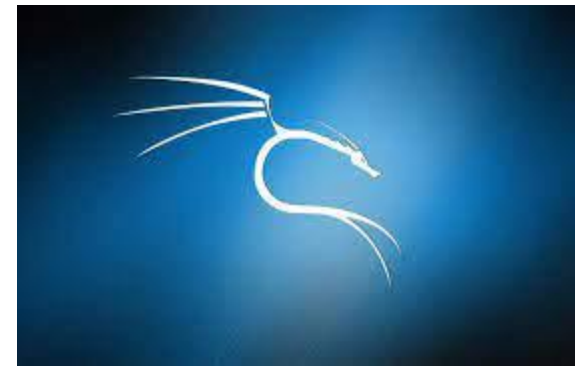
# SYSTEM SOFTWARE

System software is a collection of programs which are written to service other programs. Some system software processes complex but determinate, information structures.

Example : Operating Systems



MacOS



# APPLICATION SOFTWARE

Application software is defined as programs that solve a specific business need. Application in this area process business or technical data in a way that facilitates business operation or management technical decision making.

Example: MS Office Suite

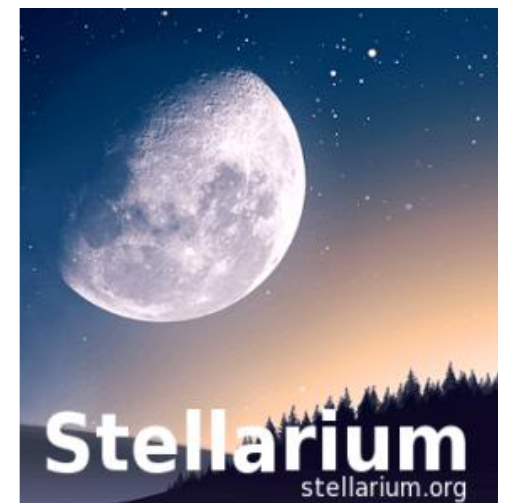


# ENGINEERING AND SCIENTIFIC SOFTWARE

This software is used to facilitate the engineering function and task.

Computer-aided design, system simulation, and other interactive applications have begun to take a real-time and even system software characteristic.

Example: MATLAB, Stellarium, ORCAD



# EMBEDDED SOFTWARE

Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself.

Fly-by-wire control systems found in aircraft.

Motion detection systems in security cameras.



# PRODUCT LINE SOFTWARE

Designed to provide a specific capability for use by many different customers,



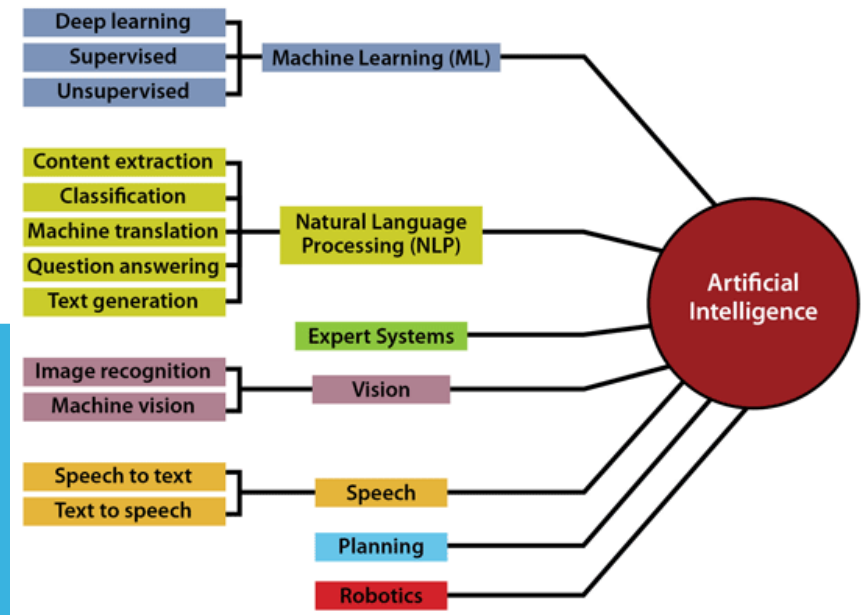
# WEB APPLICATION

It is a client-server computer program which the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics.



# AI SOFTWARE

Artificial intelligence software makes use of a non-numerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis.



### 3. SOFTWARE MYTHS

- ✓ Software Myths
- ✓ Software is easy to change
- ✓ Computers provide greater reliability than the devices they replace.
- ✓ Testing software or 'proving' software correct can remove all the errors.
- ✓ Reusing software increases safety.
- ✓ Software can work right the first time.
- ✓ Software can be designed thoroughly enough to avoid most integration problems.
- ✓ Software with more features is better software.
- ✓ Aim is to develop working programs.

# SOFTWARE MYTHS (CONTD..)

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

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

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



## 4. 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**

# 5. PROCESS FRAMEWORK

Framework is a Standard way to build and deploy applications.

Software Process Framework is a foundation of complete software engineering process. Software process framework includes all set of umbrella activities.

**Contains 5 activities :**

- Communication
- Planning
- Modelling
- Construction
- Deployment.



## 5. PROCESS FRAMEWORK CONTD.

**Under Process framework the umbrella activities include:**

- Risk Management
- Software Quality Assurance
- Software Configuration Management
- Measurement
- Format Technical Reviews



# 6.CMMI

The Capability Maturity Model Integration (CMMI) is a process and behavioral model that helps organizations streamline process improvement and encourage productive, efficient behaviors that decrease risks in software, product, and service development.

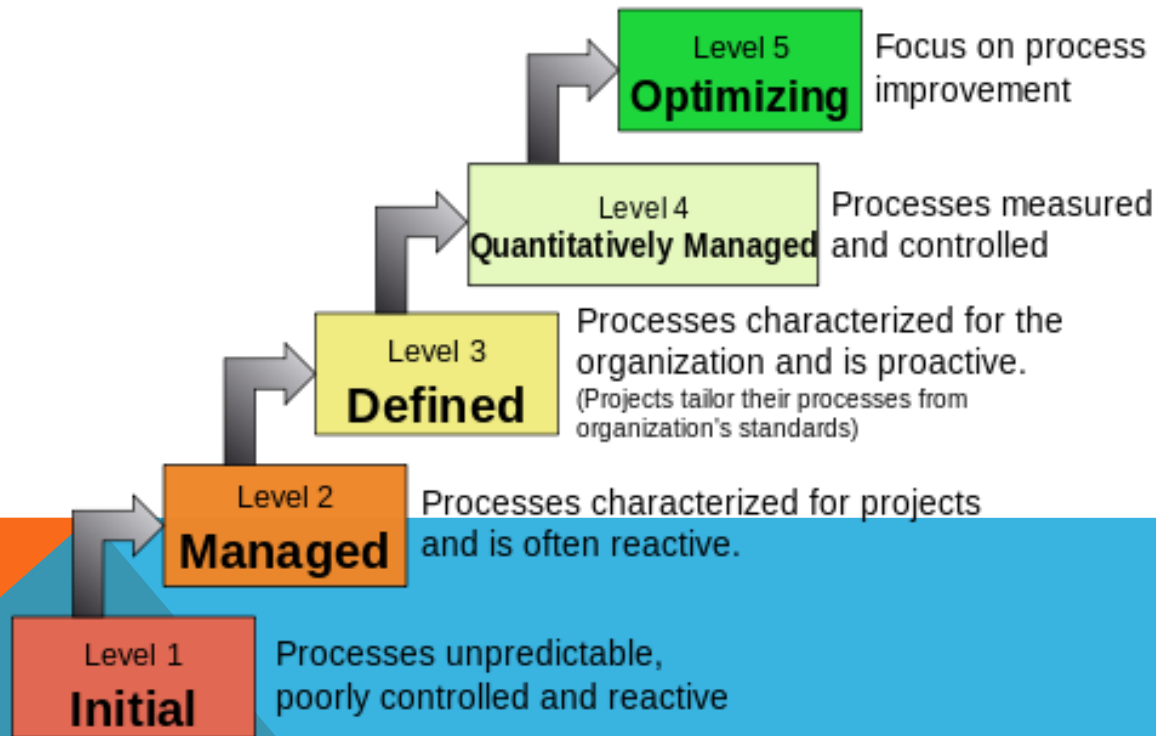
was developed by the Software Engineering Institute at Carnegie Mellon University as a process improvement tool for projects, divisions, or organizations.



**CMMI®**

# CMMI LEVELS

## Characteristics of the Maturity levels



# 7.PROCESS PATTERNS

At any level of abstraction, patterns can be defined. They can be used to describe a problem and solution associated with framework activity in some situations. While in other situations patterns can be used to describe a problem and solution associated with a complete process model.

There are 3 types of patterns :

- Stage Pattern :
  - Establishing Communication might be an example of a staged pattern
- Task Pattern:
  - Problems associated with a software engineering action or work task and relevant to successful software engineering practice
- Phase Pattern:
  - Even when the overall flow of activities is iterative in nature, it defines sequence of framework activities that occurs within process.



# PROCESS ASSESSMENT

A software process assessment is a disciplined examination of the software processes used by an organization, based on a process model.

The assessment includes the identification and characterization of current practices, identifying areas of strengths and weaknesses, and the ability of current practices to control or avoid significant causes of poor (software) quality, cost, and schedule.

There are 3 types of assessment:

- Self-assessment: Performed Internally by a company's own personnel
- Second Party Assessment: Performed by an external assessment team
- Third Party ; Performed by a third party



# 8.PERSONAL PROCESS MODELS

**Personal Software Process (PSP) is the skeleton or the structure that assist the engineers in finding a way to measure and improve the way of working to a great extend.**

**The aim of PSP is to give software engineers with the regulated methods for the betterment of personal software development processes.**

**PSP has 4 levels;**

- **Level 0 : Personal Measurement, Basic Size measures, Coding standards**
- **Level 1: Includes Planning of time and scheduling**
- **Level 2: Introduces Personal Quality Management design and code reviews.**
- **Level 3: Personal Process Evolution.**



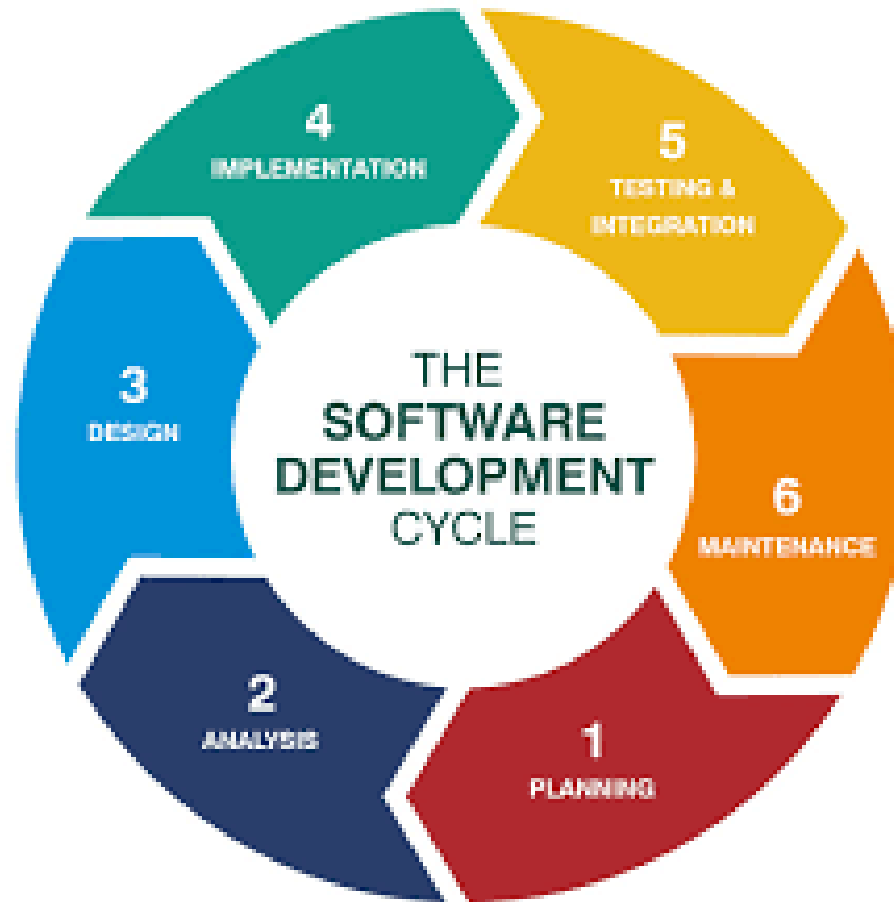
# TEAM PROCESS MODELS

The goal of TSP is to build a “self directed” project team that organizes itself to produce high quality software.

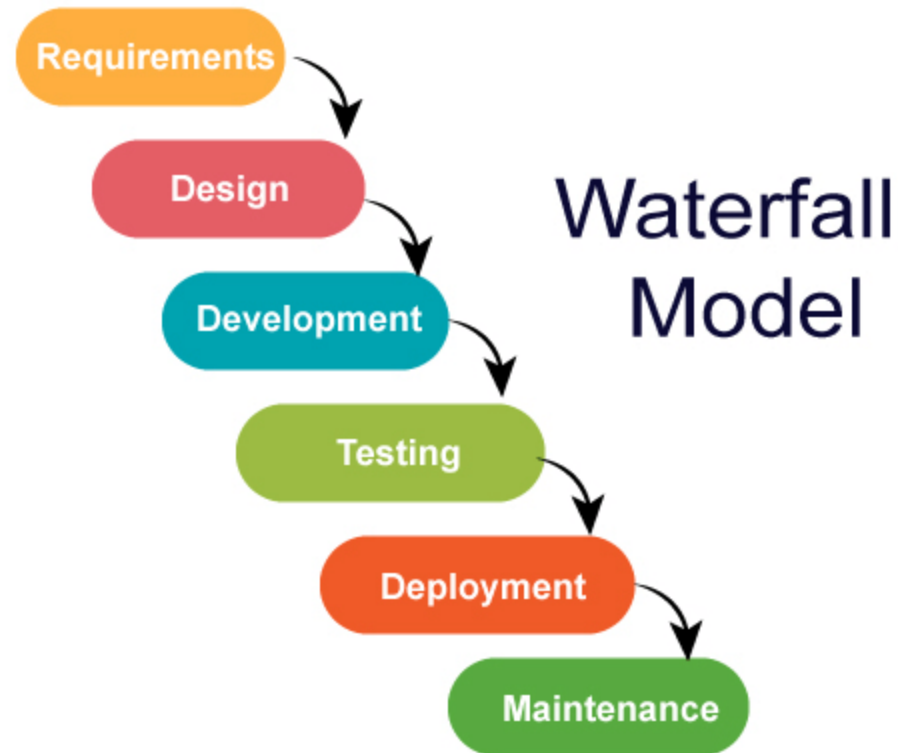
TSP defines the following framework activities: project launch, high level design, implementation, personal and team process model, integration and test, and postmortem.



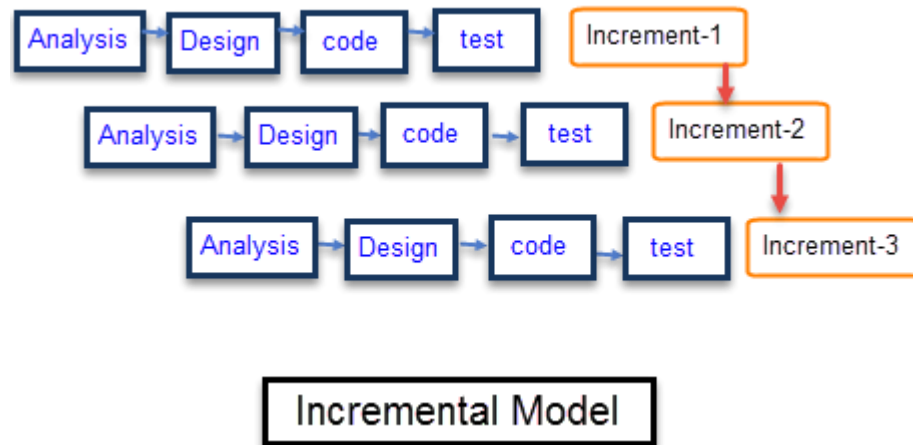
# SDLC



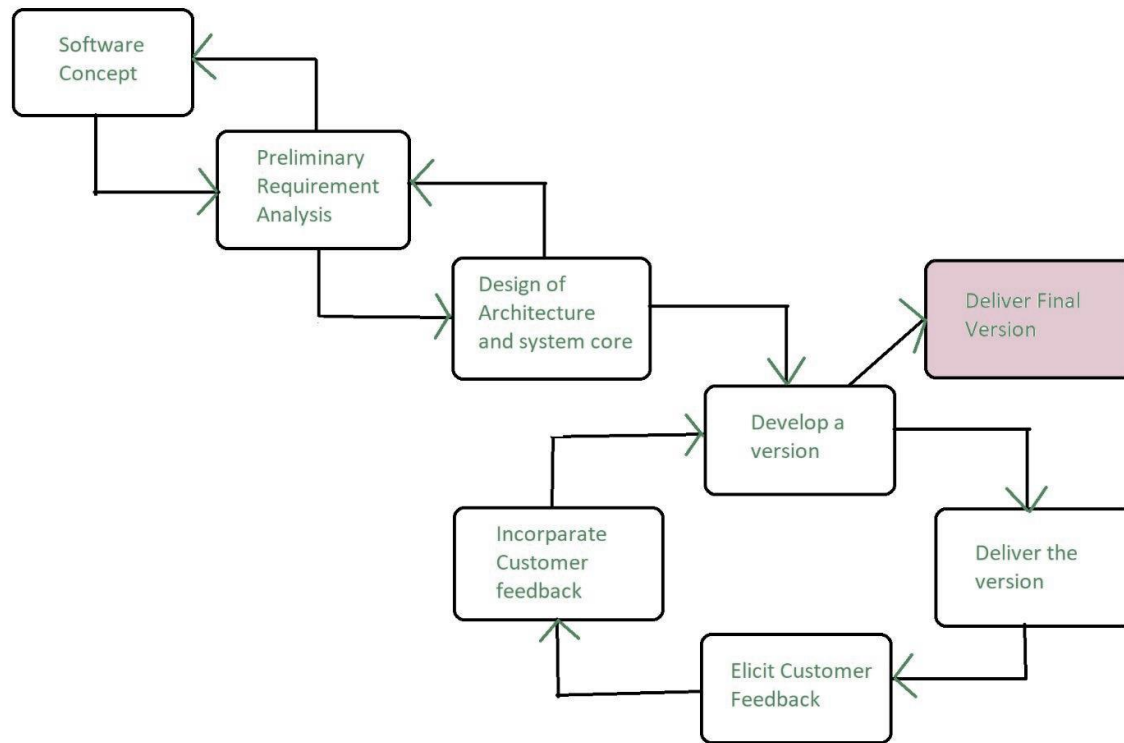
# WATERFALL MODEL



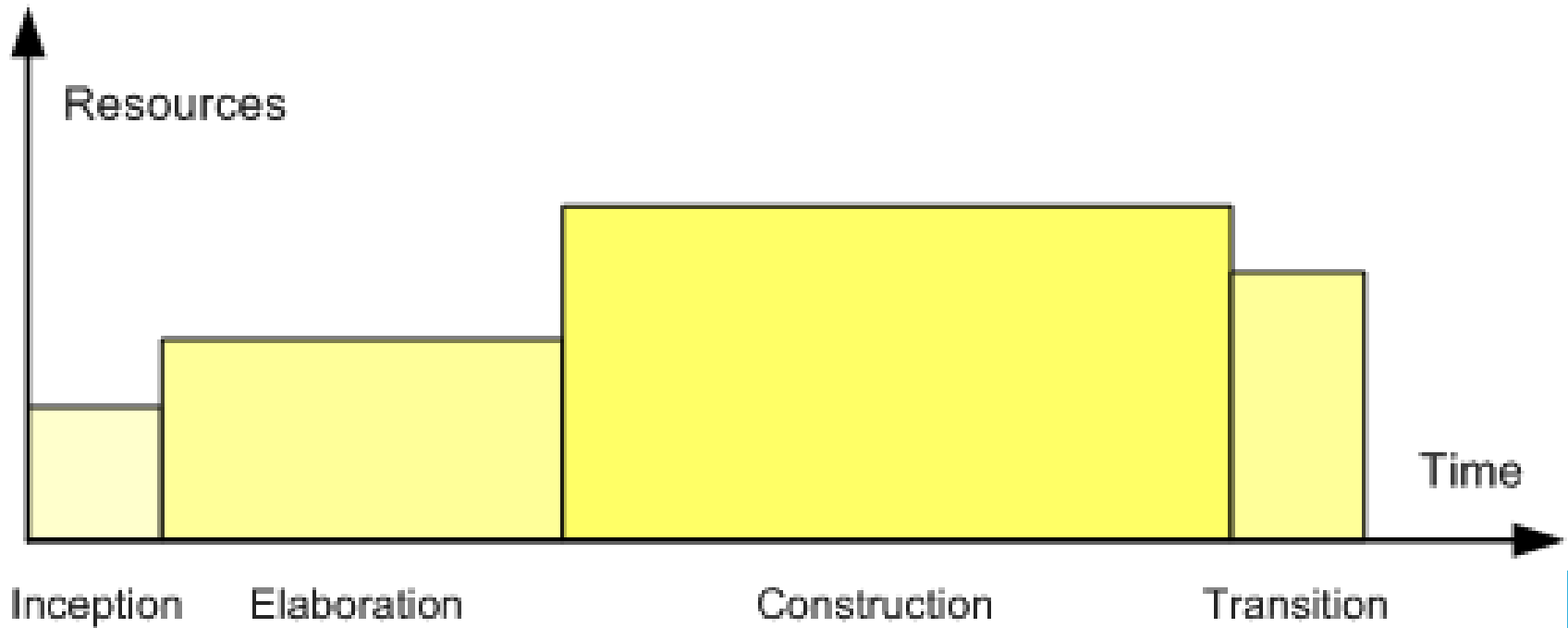
# INCREMENTAL PROCESS MODEL



# EVOLUTIONARY PROCESS MODEL

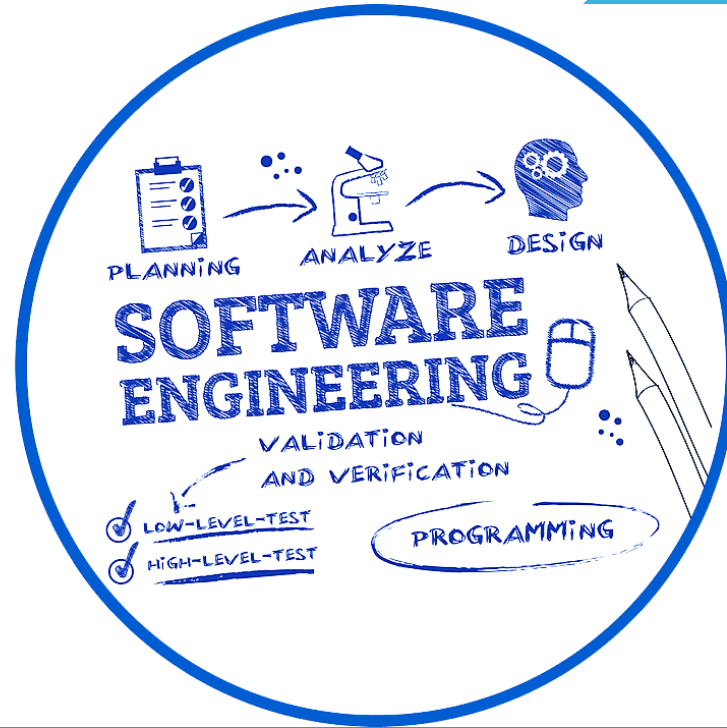


# UNIFIED PROCESS




# SOFTWARE ENGINEERING

## CHAPTER-2



# Contents-PART-I

## Software Requirements:

- Functional Requirements
  - Non-Functional Requirements
  - User Requirements
  - System Requirements
  - Interface Specification
  - Software Requirements Document
- 

# CONTENTS-PART-II

## Requirements Engineering Process:

- Feasibility Studies
- Requirements elicitation and Analysis
- Requirements validation
- Requirements Management



# CONTENTS-PART-III

## System Models :

- Context Models
- Behavioral Models
- Data Models
- Object Models
- Structured Models



## Bad Requirements

## Good Requirements



PART -I : SOFTWARE ENGINEERING :  
REQUIREMENT ENGINEERING.

# PART I

The Software requirements are a description of features and functionalities of the target system.

Characteristics:

- Clear
- Concise
- Correct
- Coherent
- Modifiable
- Verifiable
- Prioritized.

# Functional Requirements

Functional Requirements usually define if/then behaviors and include calculations, data input and business processes.

The Features Which allow the system to function as it was intended.

Examples:

- Business Rules
- Transaction Corrections
- Authentication
- Audit Tracking
- External Interfaces.



# Non-Functional Requirements

Non-Functional Requirements are the constraints or the requirements imposed on the system.

They Specify the quality attribute of the software.

Non-Functional Issues like Security, Reliability, maintenance, performance etc. are monitored

Types:

- Performance Constraints :
  - Reliability, Security, Response Time etc.
  - Operating Constraints : Physical Constraints
  - Interface Constraints : How the system is to interface with its environment.
  - Economic Constraints: Long term and Immediate Costs
  - Lifecycle Requirements : Quality of Design.



# Advantages and Disadvantages of Non-Functional Requirements.

## Advantages

They ensure the software system follows legal and adherence rules

They specify the quality attribute of the software.

They help in constructing the security policy of the software system.

## Disadvantages

The nonfunctional requirement may affect the various high-level software subsystem.

They generally increase the cost as they require special consideration during the software architecture/high-level design phase.

It is difficult to change or alter non-functional requirements once you pass them to the architecture phase.

# Functional vs Non-Functional.

Parameters	Functional Requirement	Non-Functional Requirements
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case	It is captured as a quality attribute
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software	Helps you to verify the performance of the software
Area of focus	Focuses on user requirement	Concentrates on the user's expectation and experience
Documentation	Describe what the product does	Describes how the product works
Product Info	Product features	Product properties

# User requirements

The User Requirement Specification describes the business needs for what user is expecting from the system.

It is written early in the validation process, before system is created.

Many user requirements deal with how a user will interact with a system and what that user expects

When user requirements such as these are written down, they can often break into multiple system requirements later due to switching of screens, the maximum delays in starting the process, and finally what the next screen should look like



# System Requirements

A System Requirements Specification (SRS) (also known as a Software Requirements Specification) is a document or set of documentation that describes the features and behavior of a system or software application.

In a System Requirements Document we have :

- Business Drivers
- Business Model
- Functional And System Requirement
- Business and System Use Cases
- System Qualities
- Constraints and Assumptions
- Acceptance Criteria



# Interface Specification

Interface Specification refers to the document that captures the details of Software user interface into a written document.

The Specification covers all possible actions that an end user may perform all visual auditory and other interaction elements.

There are two types of UI :

- GUI (Graphical User Interface)
- CLI(Command Line Interface)



# Software Requirements Document

A software requirements is a document that describes the intended use-case features and challenges of a software application.

These documents are created before the project has started development in order to get every stakeholder on the same page regarding the software's functionality.

The Document contains many parts : Introduction, General Description, Functional Requirements, Interface, Performance requirements, Design Constraints, Non-Functional Attributes, Appendices etc..





# SAMPLE SRS DOCUMENT

## Table of Contents for a SRS Document

### **1. Introduction**

- 1.1 Purpose
- 1.2 Document Conventions
- 1.3 Intended Audience and Reading Suggestions
- 1.4 Project Scope
- 1.5 References

### **2. Overall Description**

- 2.1 Product Perspective
- 2.2 Product Features
- 2.3 User Classes and Characteristics
- 2.4 Operating Environment
- 2.5 Design and Implementation Constraints
- 2.6 Assumptions and Dependencies

### **3. System Features**

- 3.1 Functional Requirements

### **4. External Interface Requirements**

- 4.1 User Interfaces
- 4.2 Hardware Interfaces
- 4.3 Software Interfaces
- 4.4 Communications Interfaces

### **5. Nonfunctional Requirements**

- 5.1 Performance Requirements
- 5.2 Safety Requirements
- 5.3 Security Requirements
- 5.4 Software Quality Attributes

## Bad Requirements

## Good Requirements



# Requirements Engineering Process

The Process of Defining, Documenting and Maintaining the requirements. Process of gathering and defining service provided by the system,

Consists of the following main activities :

- Requirements Elicitation
- Requirements Specification
- Requirements verification and validation
- Requirements Management.



# FEASIBILITY STUDIES

Feasibility Studies in Software Engineering is a study to evaluate the feasibility of proposed project or system.

It is one of the stage among important 4 stages of Software Project Management Process.

Carried out based on many purposes to analyze whether software product will be right in terms of development , implantation , contribution etc..

Types: Technical , Operational , Economic, Legal and Schedule



# Technical Feasibility

In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project.

This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development.

Along with this, feasibility study also analyzes technical skills and capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology



# Operational Feasibility

Degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment.

Determining suggested solution by software development team is acceptable or not

Operational Feasibility helps in taking advantage of the opportunities and fulfills the requirements as identified during the development of the project.



# Economic feasibility

Study Cost and Benefit of the Project is analyzed.

A detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on.

After that it is analyzed whether project will be beneficial in terms of finance for organization or not.



## **Legal Feasibility**

**Project is analyzed in legality point of view.**

**Analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. are done in this study.**

**To confirm if proposed project conform legal and ethical requirements.**



# Schedule Feasibility

Timelines/ deadlines are analyzed for proposed project which includes how many teams will take to complete the Project.

Includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can't be completed on time.



# Feasibility Study Process

Information Assessment

Information Collection

Report Writing

General Information



# Requirements elicitation and Analysis

Requirements Elicitation is the practice of researching and discovering the requirements of a system from users, customers and other stakeholders.

This process is also known as requirement gathering.

It is needed to know what the users really need.

This step defines what the users the need is and how the developers can develop this project.

The various methods are : Interviews, Brainstorming Sessions, Facilitated Application Specification Technique, Quality Function Deployment, Use Case Approach.



# Requirement Analysis

It is the essential activity after elicitation . We analyze, refine and scrutinize the gathered requirements to make consistent and un-ambiguous requirements.

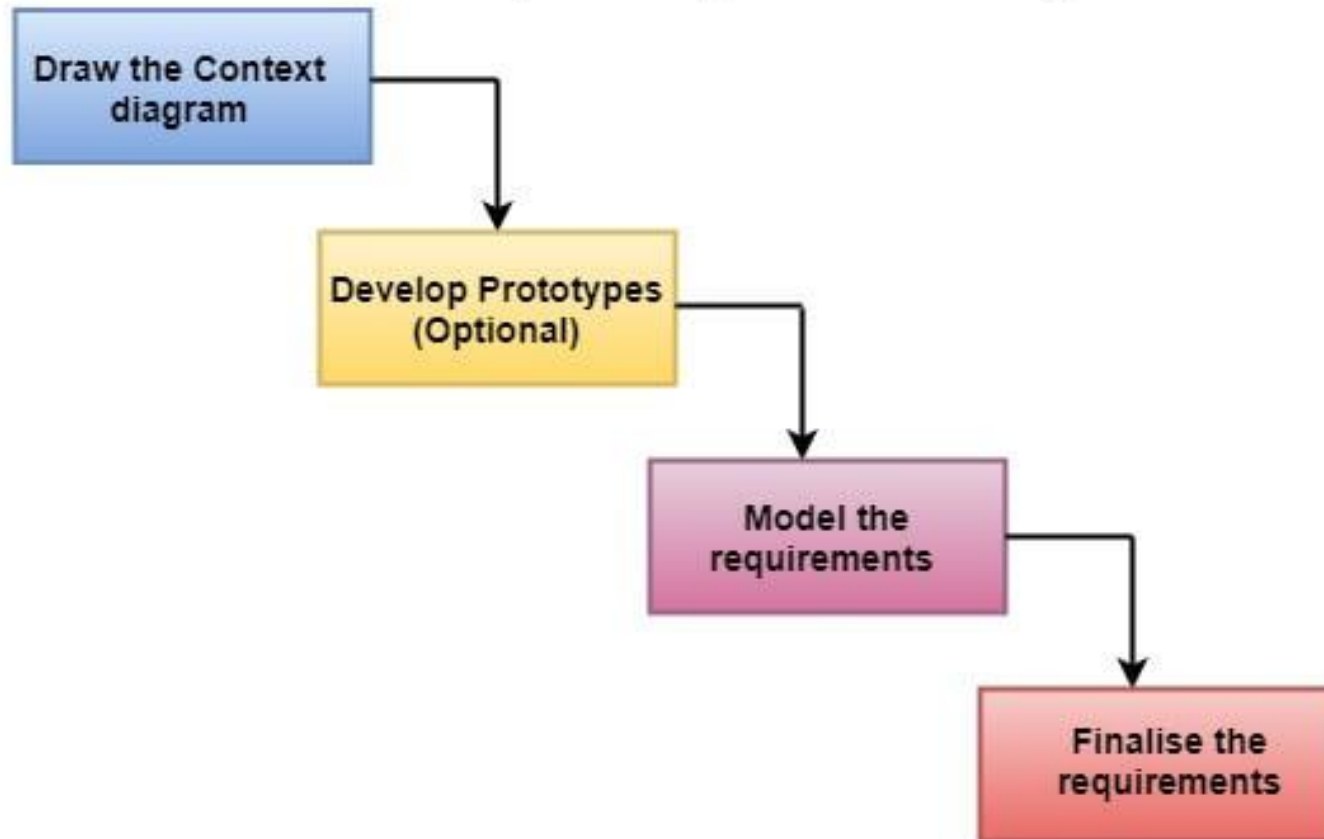
After the completion of the analysis, it is expected that the understandability of the project may improve significantly.

We may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.



# Requirement Analysis Steps

## Steps of Requirements Analysis



# Requirements Validation

Process of Checking that requirements defined for development, define the system that the customer really wants.

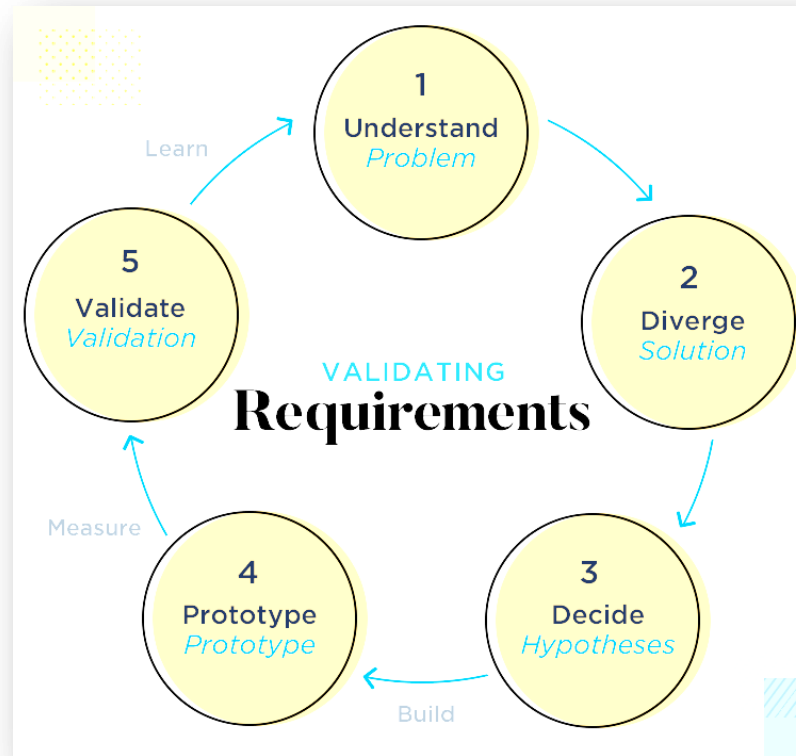
To Check issues related to requirements , we perform requirements validation.

Use this validation step to check error at the initial phase of development as the error may increase excessive rework when detected later.

These checks include: Completeness, Consistency, Validity, Realism, Ambiguity, Verifiability.



# Requirements Validation



# Requirements management

Requirement management is the process of analyzing , documenting tracking prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders.

This stake takes care of the changing nature of requirements.

Should be ensured that SRS is as modifiable as possible so as to incorporate changes in requirements specified in a systematic and controlled manner is an important part of the requirements process.



# Requirements Management



# Chapter 2

PART-III: SYSTEM

SYSTEMS  
MODELS

Assumptions  
Simplification  
Limitations  
Constraints  
Preferences

# SYSTEM MODEL & SYSTEM MODELLING

The interdisciplinary study of the use of models to conceptualize and construct systems in businesses and IT development.

System Modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

For System Modelling we use the concept of data models to understand.

We have the following models : Context, Behavioral, Data, Object and Structured.



# CONTEXT MODEL

Defines how context data are structured and maintained.

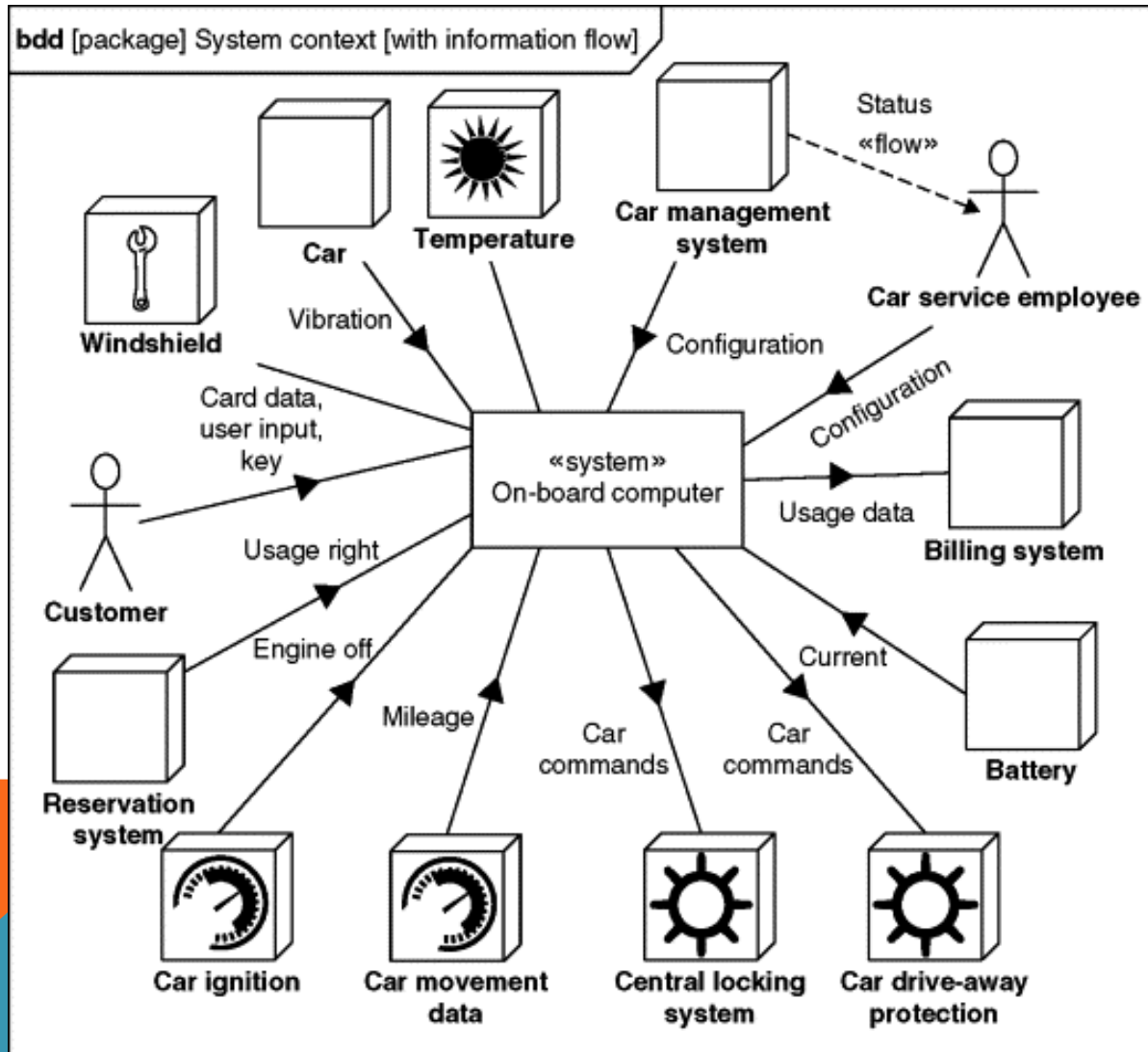
A key role of context model is to simplify and introduce greater structure into the task of developing context aware applications.

A key role of developing a context model is to simplify and introduce a greater structure into the task of developing a context aware applications.

Best Example: Unified Modelling Language as used in systems engineering defines a context model as the physical scope of the system being designed.



# CONTEXT MODEL



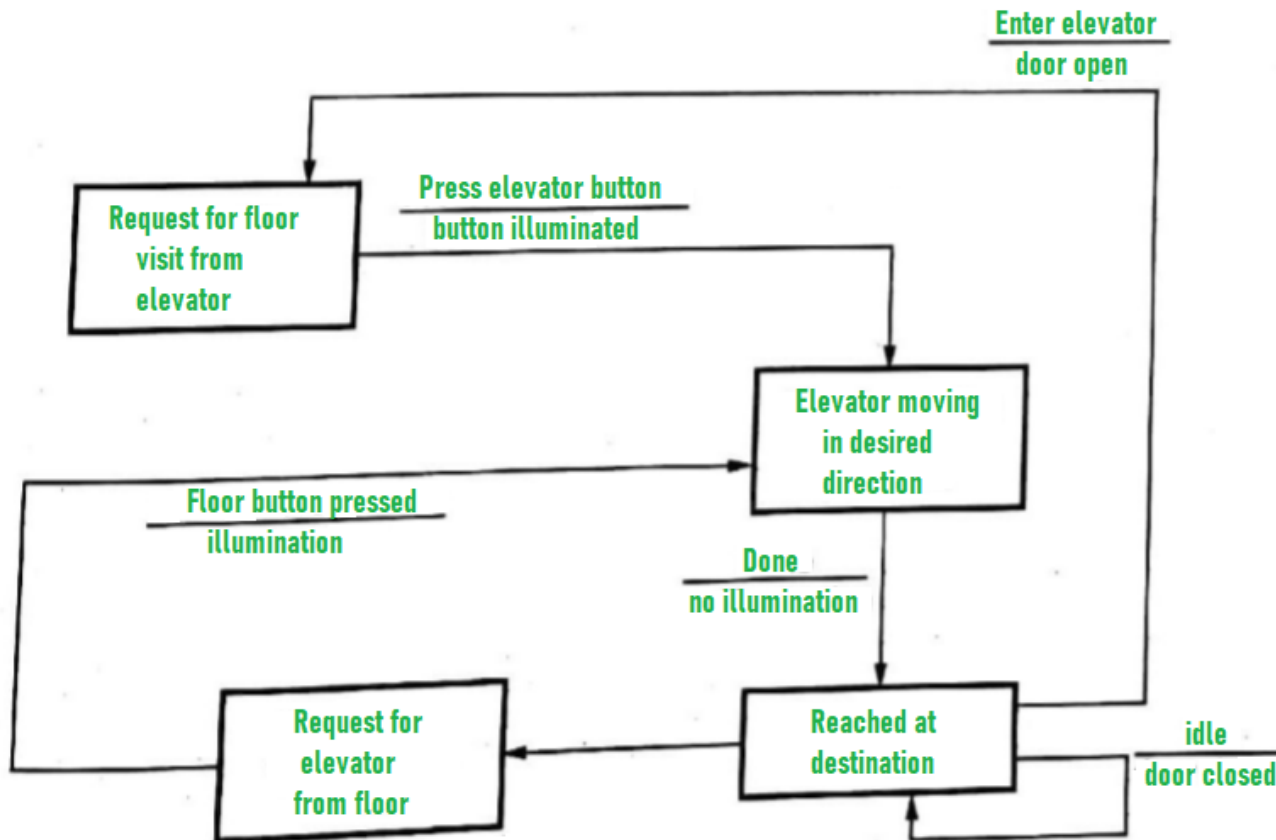
# Behavioral Model

Behavioral Model is specially designed to make us understand behavior and factors that influence behavior of a System. Behavior of a system is explained and represented with the help of a diagram. This diagram is known as State Transition Diagram. It is a collection of states and events. It usually describes overall states that a system can have and events which are responsible for a change in state of a system.

So, on some occurrence of a particular event, an action is taken and what action needs to be taken is represented by State Transition Diagram.



# Behavioral Model



**STATE TRANSITION DIAGRAM**

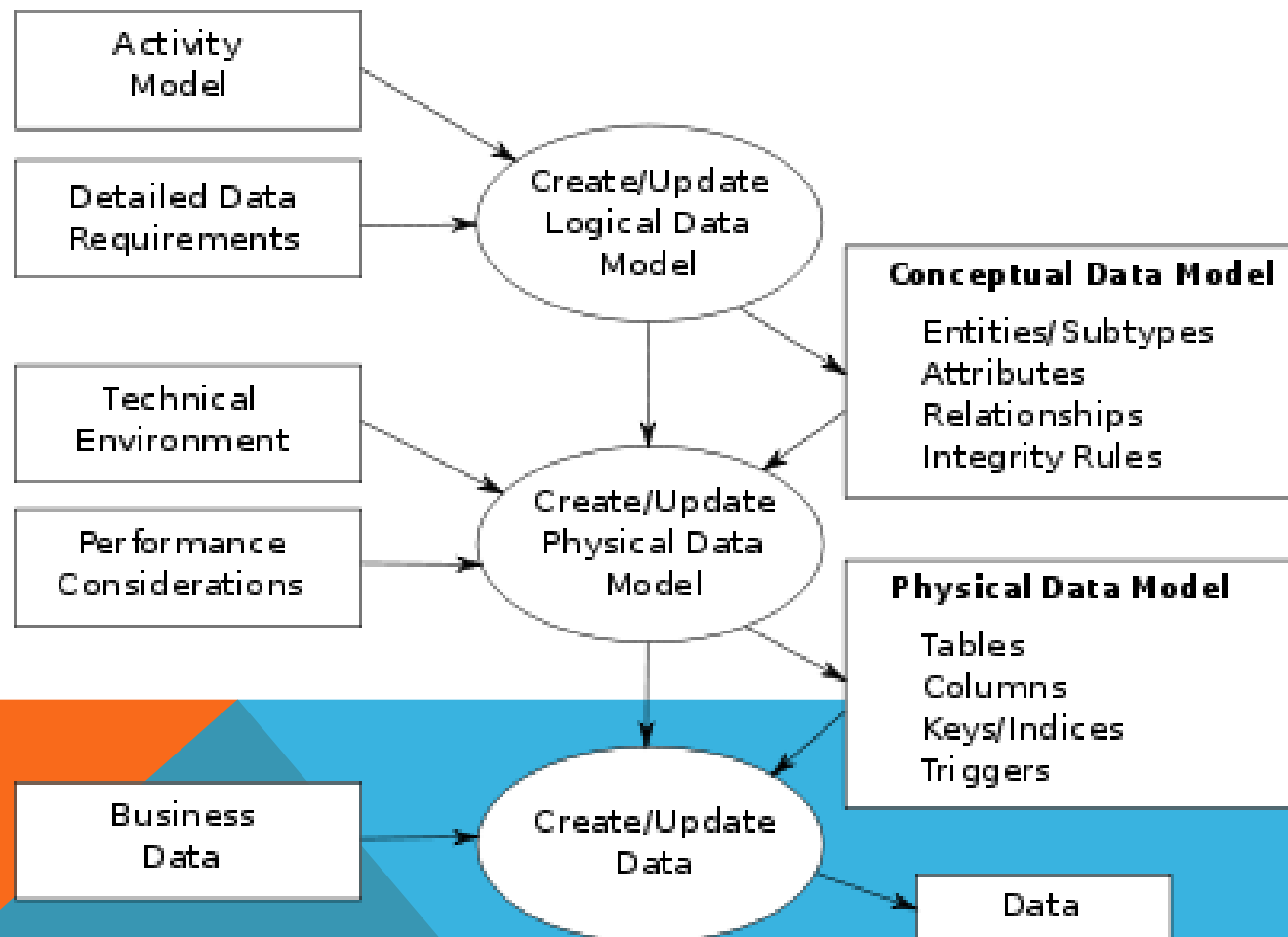
# DATA MODEL

Data modeling in software engineering is the process of creating a data model by applying formal data model descriptions using data modeling techniques. Data modeling is a technique for defining business requirements for a database.

The goal is to create a visual data map that accurately describes the data structure, how data will flow through the system whilst highlighting important data relationships. This can involve the data input itself, the data infrastructure and output, whether that's predictive models, ML algorithms, AI or other products/services.



# DATA MODEL



# OBJECT MODEL

Object Modeling Technique (OMT) is real world based modeling approach for software modeling and designing.

It was developed basically as a method to develop object-oriented systems and to support object-oriented programming.

It describes the static structure of the system.

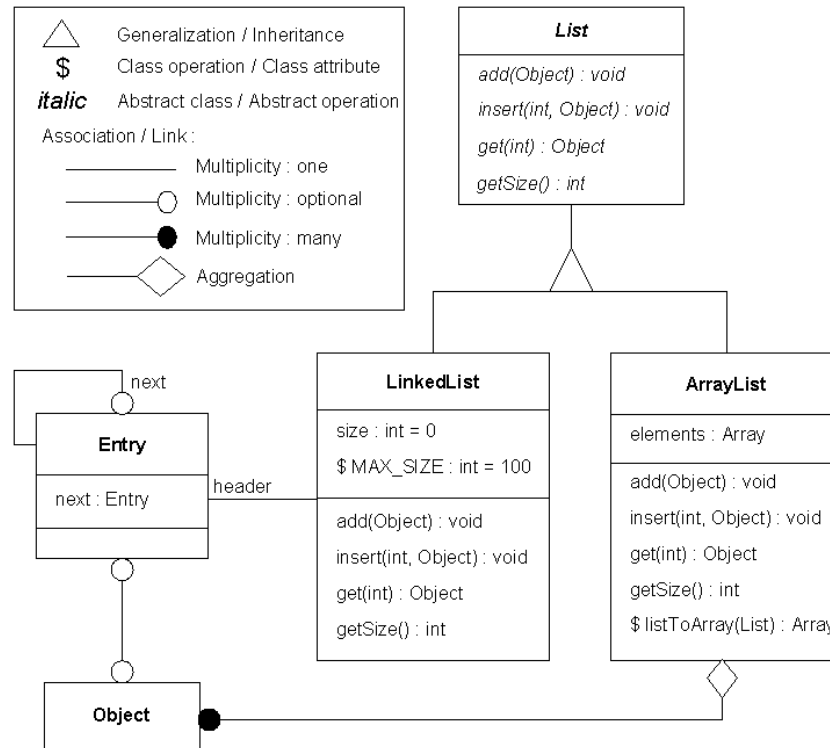
Object Modeling Technique is easy to draw and use

OMT is one of the most popular object oriented development techniques used now-a-days.

OMT was developed by *James Rumbaugh*.



# OBJECT MODEL



# STRUCTURED MODEL

Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.

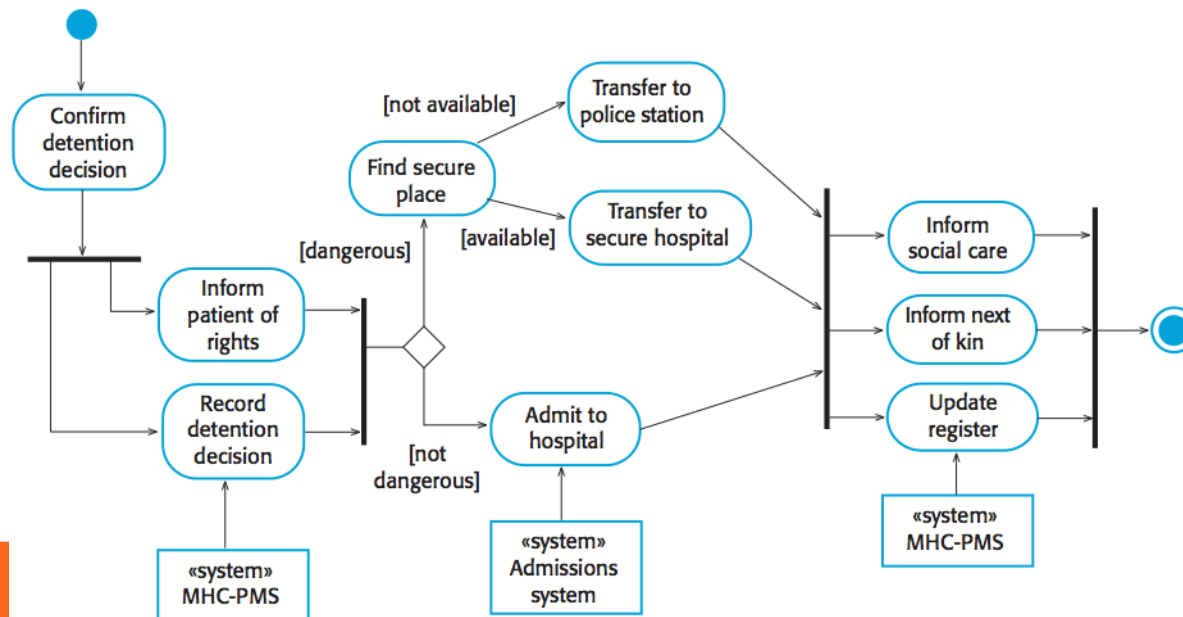
Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.

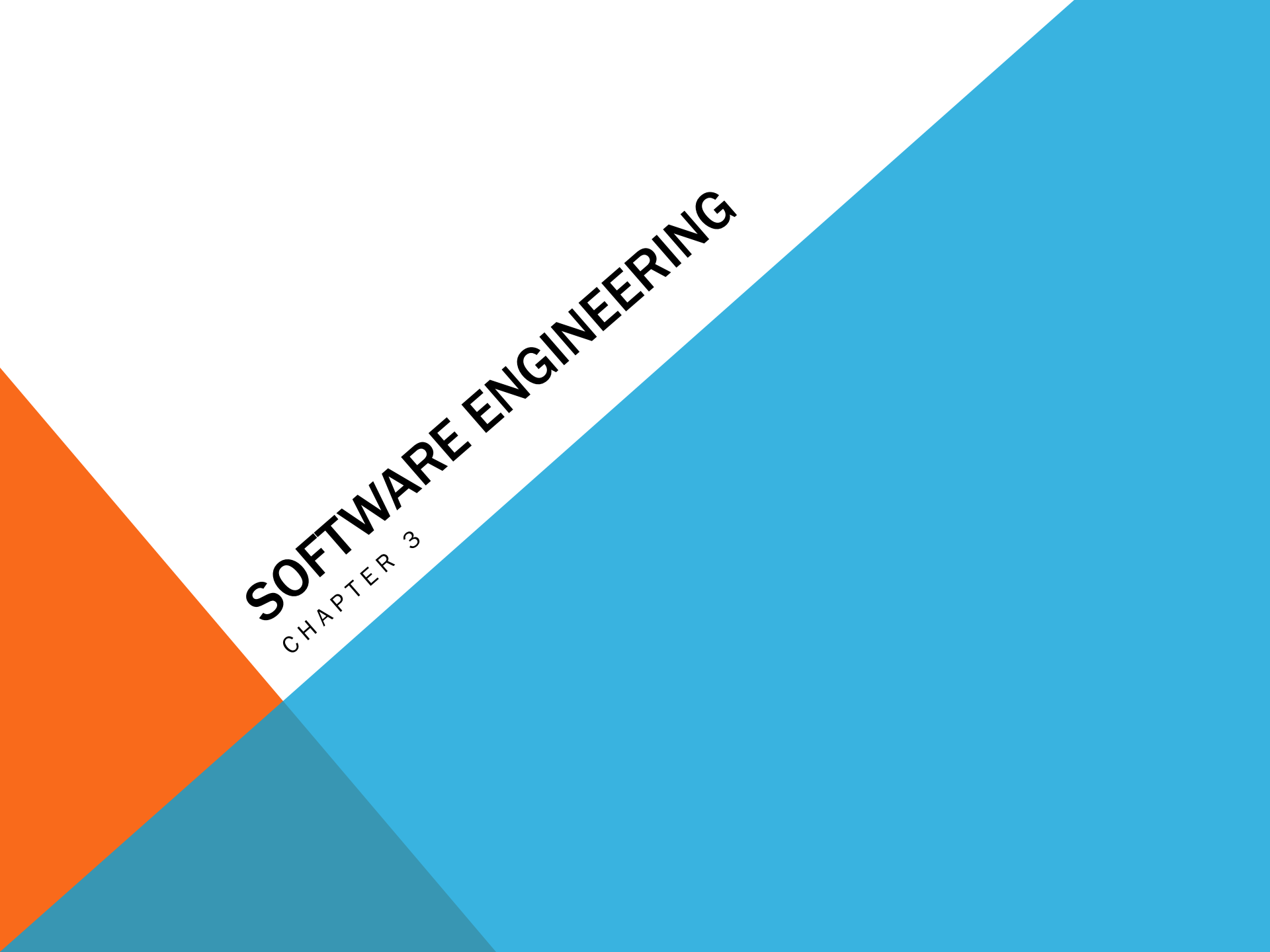
Structural models show the organization and architecture of a system.

Class diagrams are used to define the static structure of classes in a system and their associations.



# STRUCTURED MODEL





# **SOFTWARE ENGINEERING**

CHAPTER 3

# CONTENTS

## Design Engineering :

- Design Engineering Definition
- Process and Quality
- Design Concepts
- The Design Model.



# CONTENTS PART II

Software Architecture

Data Design

Architectural Styles and Patterns

Architectural Design

Conceptual Model

Basic Structural Modelling

Class Diagrams

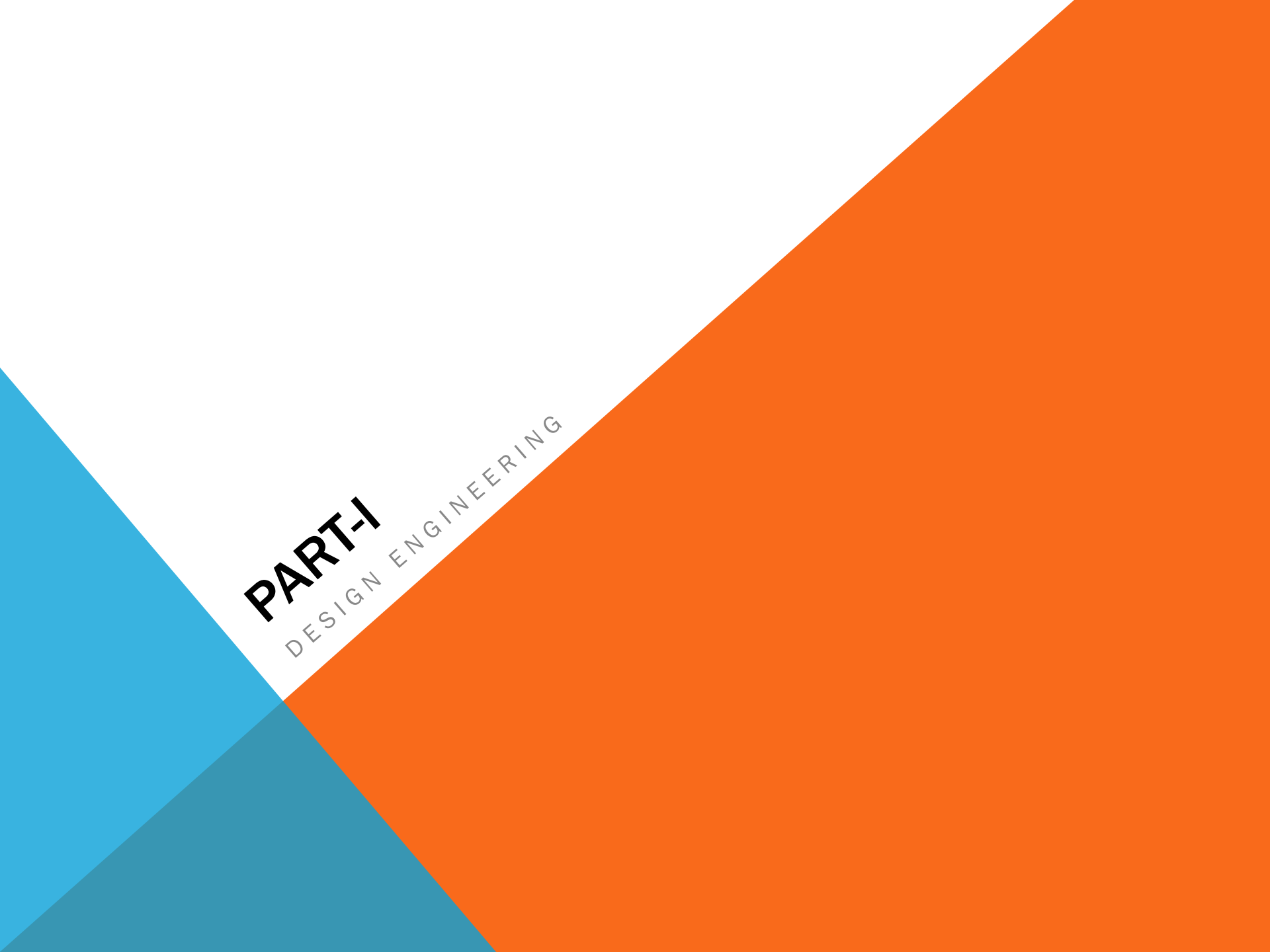
Sequence diagrams

Collaboration diagrams

Use case diagrams

Component Diagrams





# **PART-I**

DESIGN ENGINEERING

# DESIGN ENGINEERING

The Design Phase of Software Engineering deals with transforming the customer requirements as described in the SRS.

The Design Process can be divided into 3 parts :

- Interface Design
- Architectural Design
- Detailed Design



# INTERFACE DESIGN

- *Interface design* is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored and the system is treated as a black box.
- Interface design should include the following details:
  - Precise description of events in the environment, or messages from agents to which the system must respond.
  - Precise description of the events or messages that the system must produce.
  - Specification on the data, and the formats of the data coming into and going out of the system.

# INTERFACE DESIGN

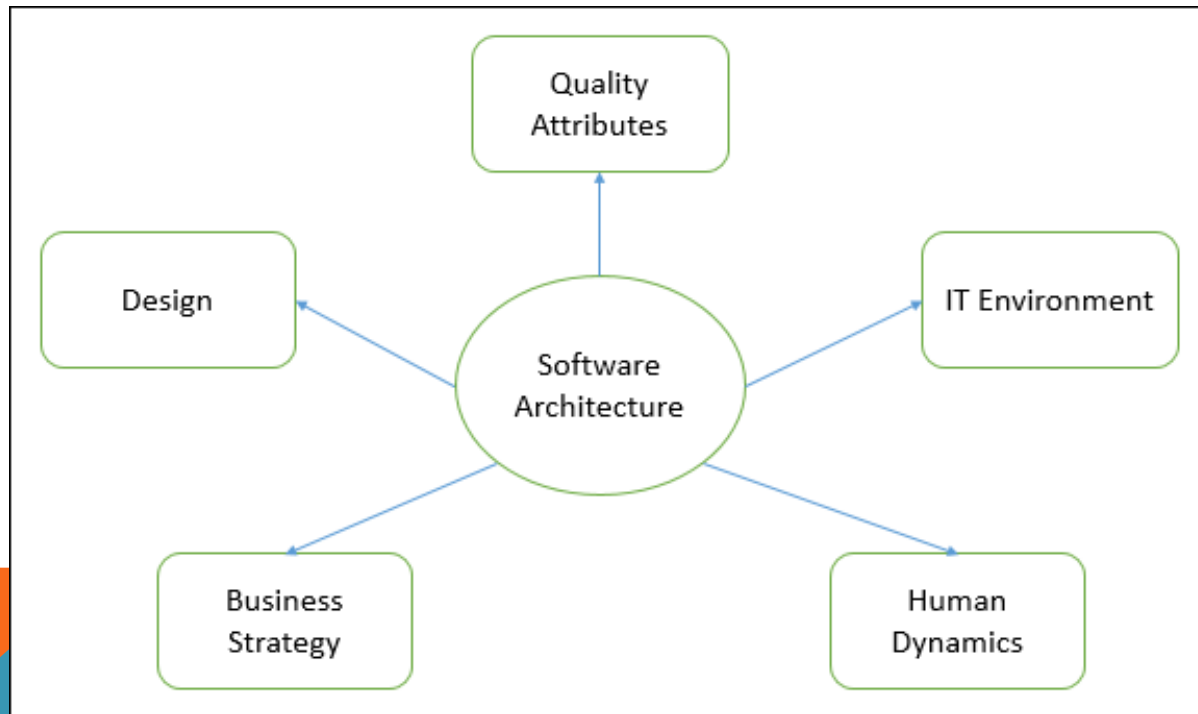


# ARCHITECTURAL DESIGN

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



# ARCHITECTURAL DESIGN



# DETAILED DESIGN

- *Design* is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.
- The detailed design may include: User interfaces, Unit states and state changes, Data and control interaction between units, Algorithms and data structures etc.

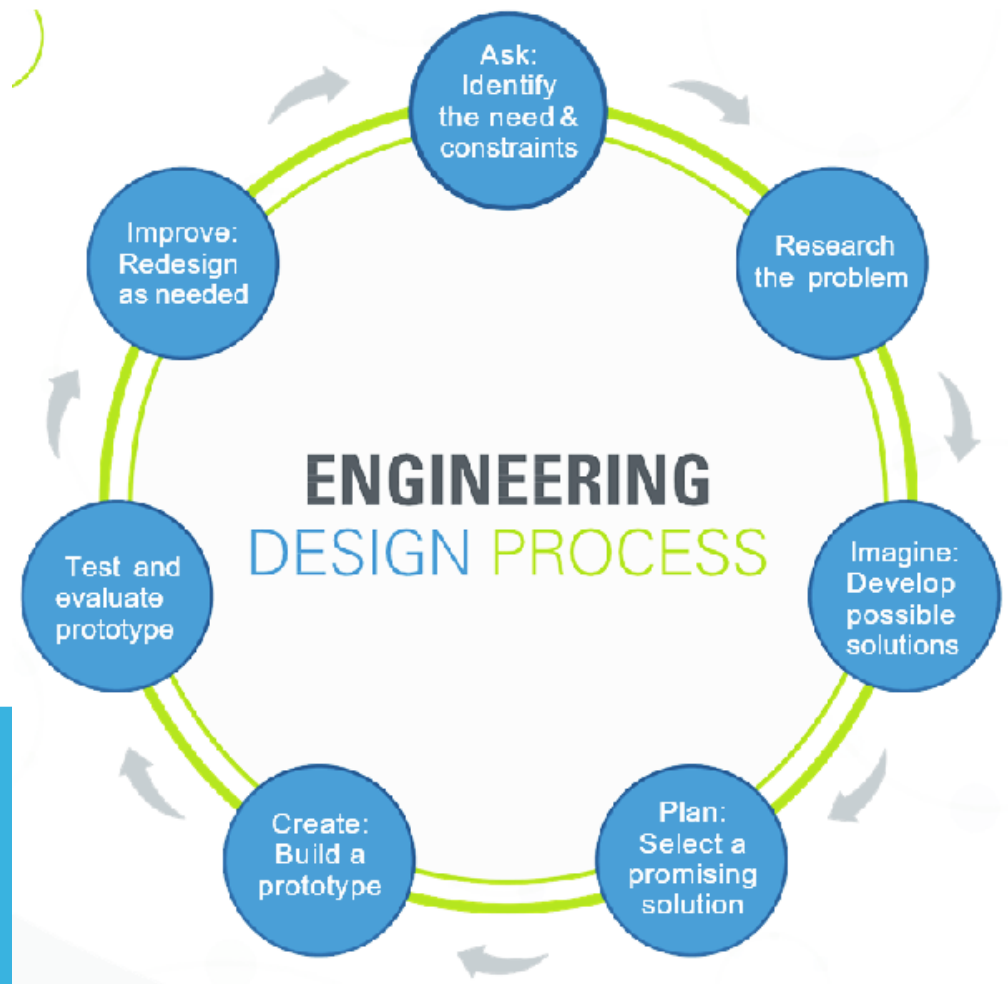


# DESIGN ENGINEERING PROCESS

- The engineering design process is a series of steps that engineers follow to find a solution to a problem. The steps include problem solving processes such as, for example, determining your objectives and constraints, prototyping, testing and evaluation.
- While the design process is iterative it follows a predetermined set of steps, some of these may need to be repeated before moving to the next one.



# DESIGN ENGINEERING PROCESS DIAGRAM



# STEPS OF DESIGN ENGINEERING

## STEP-1: DEFINE THE PROBLEM

- What is the problem or need?
- Who has the problem or need?
- Why is it important to solve?



# STEPS OF DESIGN ENGINEERING

## STEP 2: DO BACKGROUND RESEARCH

- Learn from the experiences of others — this can help you find out about existing solutions to similar problems, and avoid mistakes that were made in the past. So, for an engineering design project, do background research in two major areas:
- Users or customers
- Existing solutions



# STEPS IN DESIGN ENGINEERING

## STEP 3: SPECIFY REQUIREMENTS

Design requirements state the important characteristics that your solution must meet to succeed. One of the best ways to identify the design requirements for your solution is to analyze the concrete example of a similar, existing product, noting each of its key features.



# STEPS IN DESIGN ENGINEERING

## STEP-4: BRAINSTORM SOLUTIONS

There are always many good possibilities for solving design problems. If you focus on just one before looking at the alternatives, it is almost certain that you are overlooking a better solution. Good designers try to generate as many possible solutions as they can.



# **STEPS IN DESIGN ENGINEERING**

## **STEP 5: CHOOSE THE BEST SOLUTION**

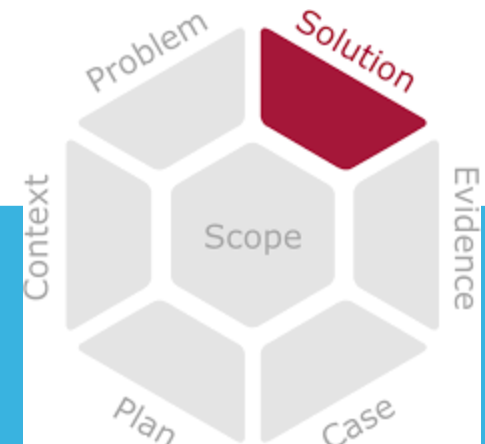
**Look at whether each possible solution meets your design requirements. Some solutions probably meet more requirements than others. Reject solutions that do not meet the requirements.**



# STEPS OF DESIGN ENGINEERING

## STEP-6: DEVELOP THE SOLUTION

- Development involves the refinement and improvement of a solution, and it continues throughout the design process, often even after a product ships to customers.



# STEPS OF DESIGN ENGINEERING

## STEP-7: BUILD A PROTOTYPE

A prototype is an operating version of a solution.

Often it is made with different materials than the final version, and generally it is not as polished.

Prototypes are a key step in the development of a final solution, allowing the designer to test how the solution will work.

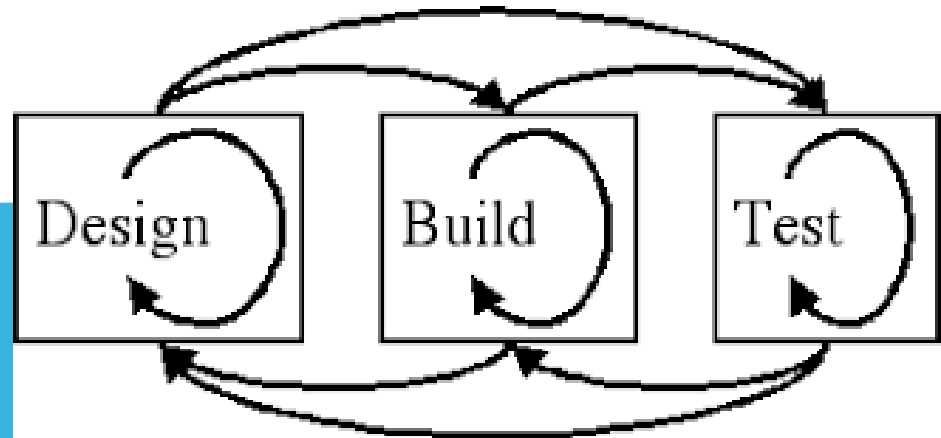


# STEPS OF DESIGN ENGINEERING

## STEP-8: TEST AND RE-DESIGN

The design process involves multiple iterations and redesigns of your final solution.

You will likely test your solution, find new problems, make changes, and test new solutions before settling on a final design.



# STEPS OF DESIGN ENGINEERING

## STEP-COMMUNICATE RESULTS

- To complete your project, communicate your results to others in a final report and/or a display board.
- Professional engineers always do the same, thoroughly documenting their solutions so that they can be manufactured and supported.



# DESIGN ENGINEERING QUALITY

- Quality engineering is the discipline of engineering concerned with the principles and practice of product and service quality assurance and control.
- In software development, it is the management, development, operation and maintenance of IT systems and enterprise architectures with a high quality standard.



# DESIGN CONCEPTS

A set of the Concepts that go hand in hand along with the design engineering of the software system are known as Design Concepts

These Concepts are a key factor in analyzing the design of the software.

There are Few Concepts that are categorized as Design Concepts



# 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.



### **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.



# 6.1 COHESION

Cohesion is an extension of the information hiding concept.

A cohesive module performs a single task

It requires a small interaction with the other components in other parts of the program.



## 6.2 COUPLING

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

More precisely it is the interdependence between software modules.

Or it can be a measure of how closely connected two modules are.



# 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 behavior.

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



## 9. DESIGN CLASSES

- The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible.



# THE DESIGN MODEL

- Design modeling in software engineering represents the features of the software that helps engineer to develop it effectively, the architecture, the user interface, and the component level detail.
- Different methods like data-driven, pattern-driven, or object-oriented methods are used for constructing the design model.
- All these methods use set of design principles for designing a model.



# DESIGN MODELLING

- The design model builds on the analysis model by describing, in greater detail, the structure of the system and how the system will be implemented.
- In the design model, packages contain the design elements of the system, such as design classes, interfaces, and design subsystems, that evolve from the analysis classes each package can contain any number of sub-packages that further partition the contained design elements.
- These architectural layers form the basis for a second-level organization of the elements that describe the specifications



# **SOFTWARE ENGINEERING**

## **CHAPTER 3**

PART 2

# SOFTWARE ARCHITECTURE

- Software architecture refers to the fundamental structures of a software system and the discipline of creating such structures and systems.
- The architecture of the system is a metaphor analogous to the architecture of a building.
- Functions as a blueprint for the development of the software.



# SOFTWARE ARCHITECTURE TYPES

The most used software architectures are:

- Business Architecture
- Application Architecture
- Information Architecture
- Information Technology Architecture



# BUSINESS ARCHITECTURE

Business architecture defines the strategy of business, governance, organization and key business processes within an enterprise.

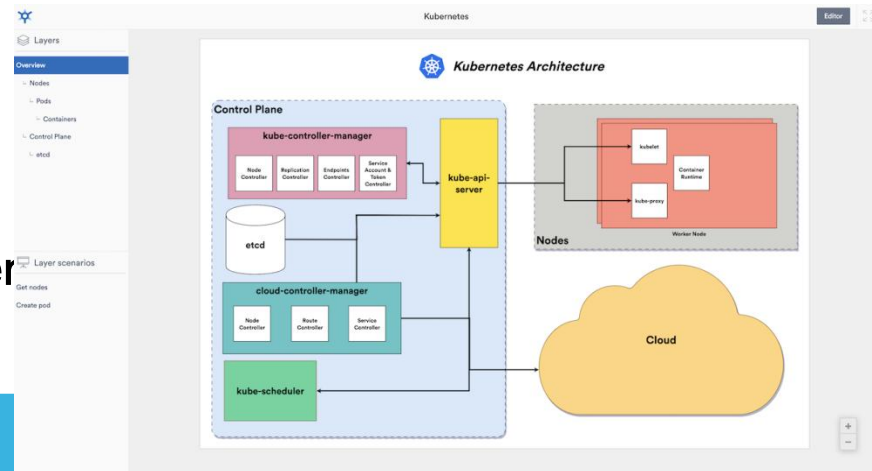
This type of architecture focuses on the analysis and design of business processes.



# APPLICATION ARCHITECTURE

- It describes the patterns and techniques used to design and build an application.
- Gives a roadmap and best practices to follow when building an application.

- This diagram is a representation



# INFORMATION ARCHITECTURE

- It is structural design of shared information environments; the art and science of organizing and labelling websites, intranets, online communities and software to support usability and findability.
- A discipline that focuses on the organization of information within digital products.



# INFORMATION TECHNOLOGY ARCHITECTURE

Process of development of methodical information technology specifications, models and guidelines.

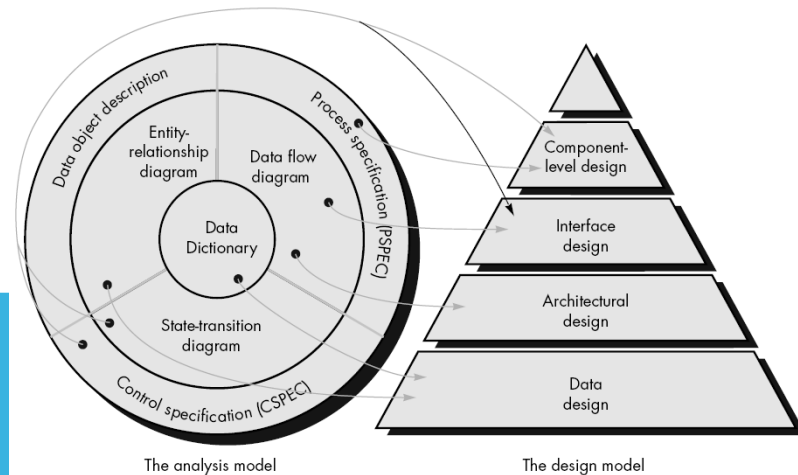
It uses a variety of Information Technology notations for example UML within a coherent information architecture.

Focuses on three basic tiers within organization.



# DATA DESIGN

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



Translating the analysis model into a software design

# ARCHITECTURAL STYLE

Shows how we organize our code or how the system will look like from an aerial view.

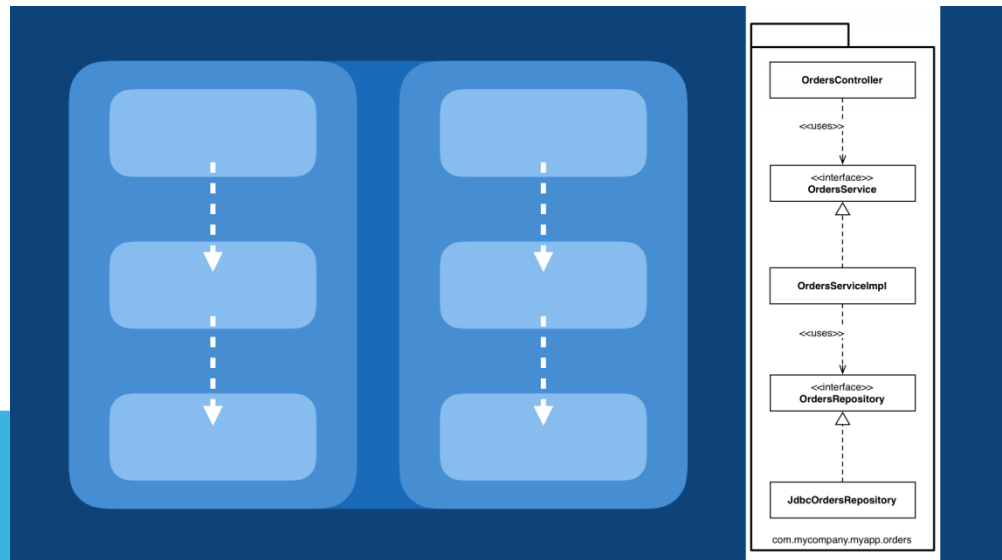
The list of style:

- Structure Architecture Style
- Messaging Styles
- Distributed Systems
- Shared memory Styles
- Adaptive System Style.



# STRUCTURE ARCHITECTURE STYLE

- It consists of Several Component based Styles such as :
  - Layers
  - Pipes
  - Filters



# MESSAGING STYLES

Messaging refers to various forms of communication information to users

Some of the forms are : E-Mail , SMS, EMS, MMS , Instant Messaging , HDML  
Notifications , WAP Push

Based on the software requirement, developer can design the messaging system.

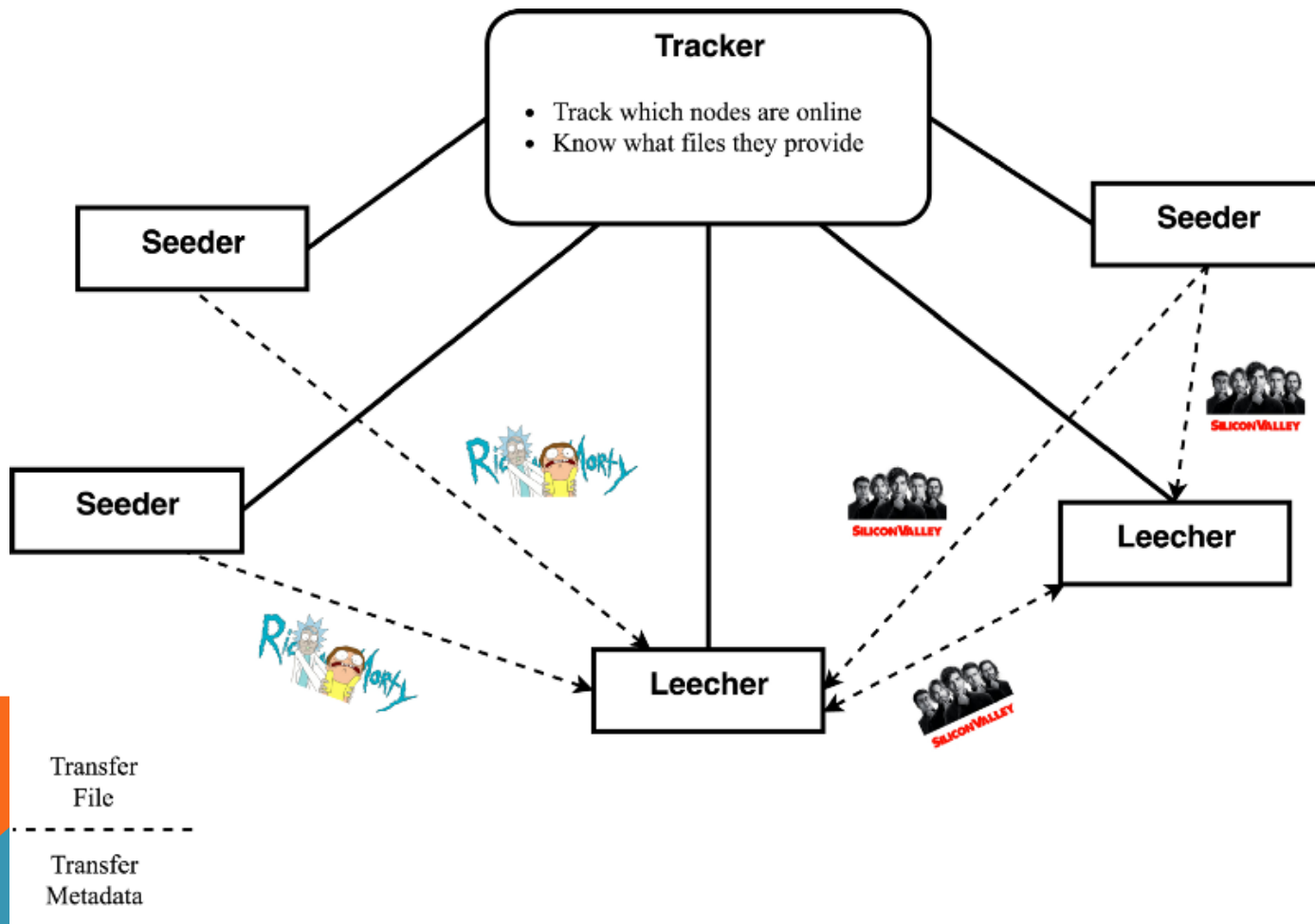


# DISTRIBUTED SYSTEMS

- A distributed computer system consists of multiple software components that are on multiple computers, but run as a single system.
- The computers that are in a distributed system can be physically close together and connected by a local network, or they can be geographically distant and connected by a wide area network.



# DISTRIBUTED SYSTEM



# SHARED MEMORY STYLES

- Shared Memory Consists of three types:

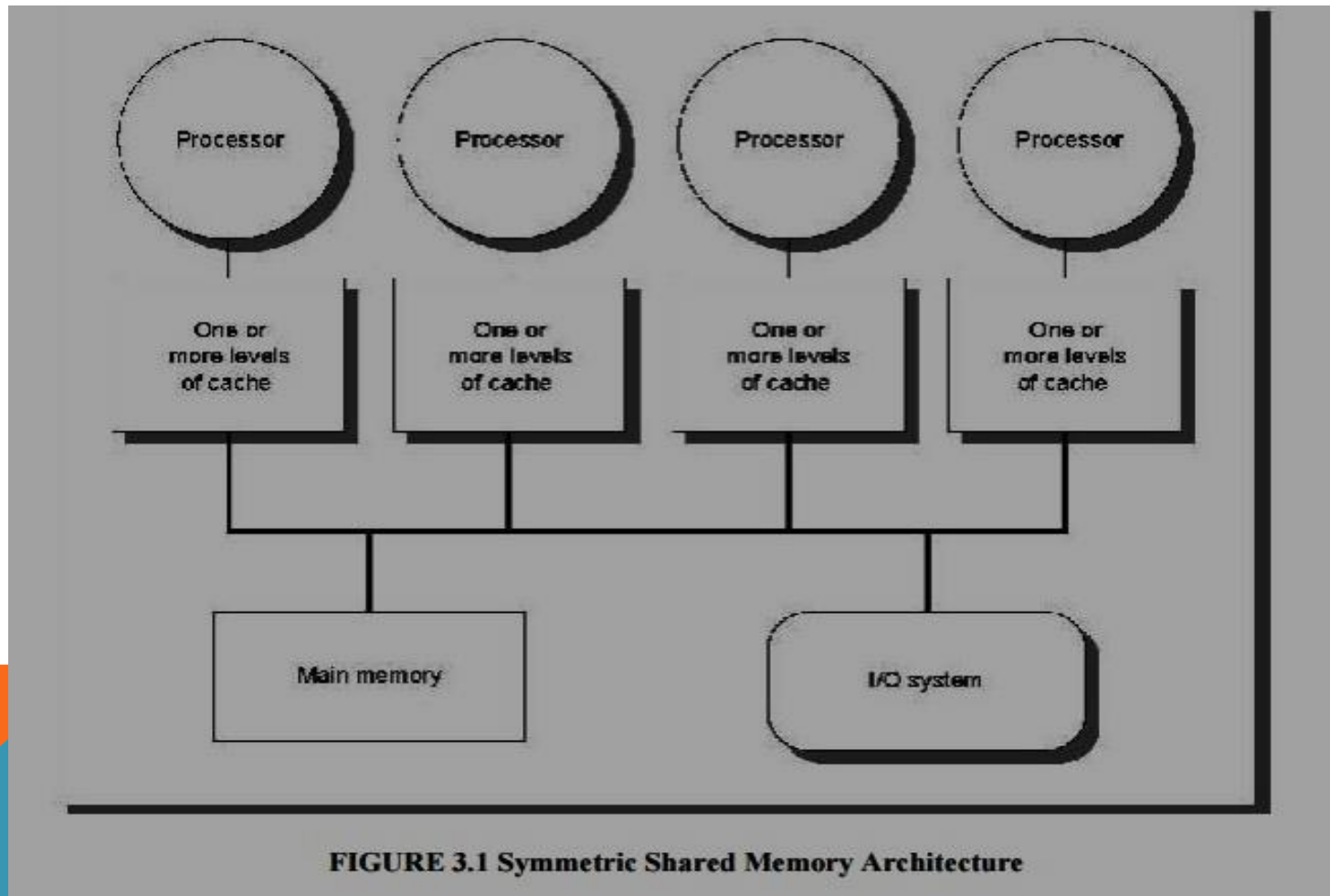
Database Centric : Based on a DB

Blackboard: An AI Approach

Rule based: Applicable in most systems where automatic rule inference re executed.



# SHARED MEMORY

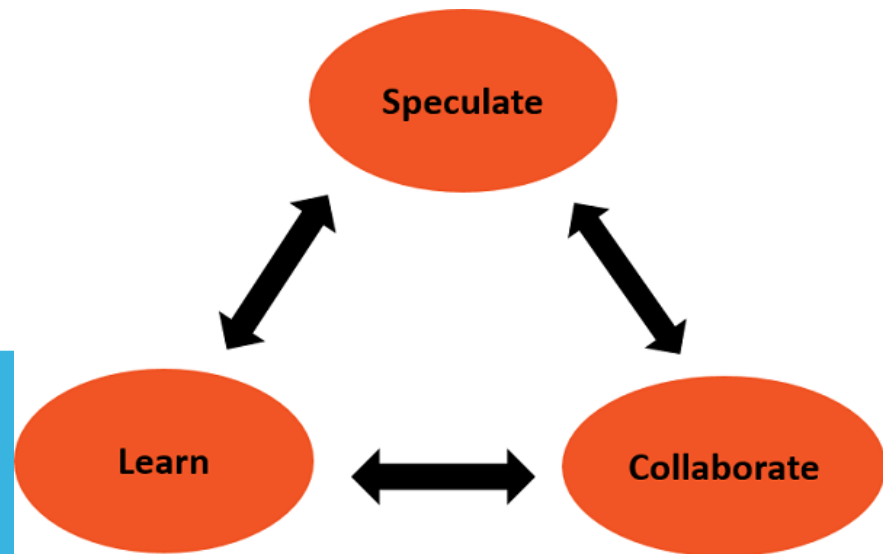


# ADAPTIVE SYSTEM STYLES

These Styles consists of Microkernel Style , reflection , domain Specific language styles

.

The System that changes its behavior in response to its environment.



# ARCHITECTURAL PATTERNS

- Architectural Design Pattern are accumulative best practices and experiences that software professionals used over the years to solve the general problem by – trial and error – they faced during software development.
  - Two main principles of object-oriented design:
    - Develop to an interface, not to an implementation.
    - Favor object composition over inheritance.
- They are Creational Patterns, Structural Patterns, Behavioral Patterns.



# CREATIONAL DESIGN PATTERN

- Provide a way to create objects while hiding the creation logic. Thus, the object creation is to be done without instantiating objects directly with the “New” keyword to gives the flexibility to decide which objects need to be created for a given use case.
- Abstract Factory Pattern, Singleton Pattern, Builder Pattern and Prototype Pattern.



# STRUCTURAL PATTERN

- These Patterns are concerned with class and object composition of the system
- Adapter, Bridge, Filter, Composite, Decorator, Façade, Flyweight and Proxy Come under these Structural Pattern.



# BEHAVIORAL PATTERN

Behavioral Patterns are concerned with communications between objects.

The communication can be of any type.

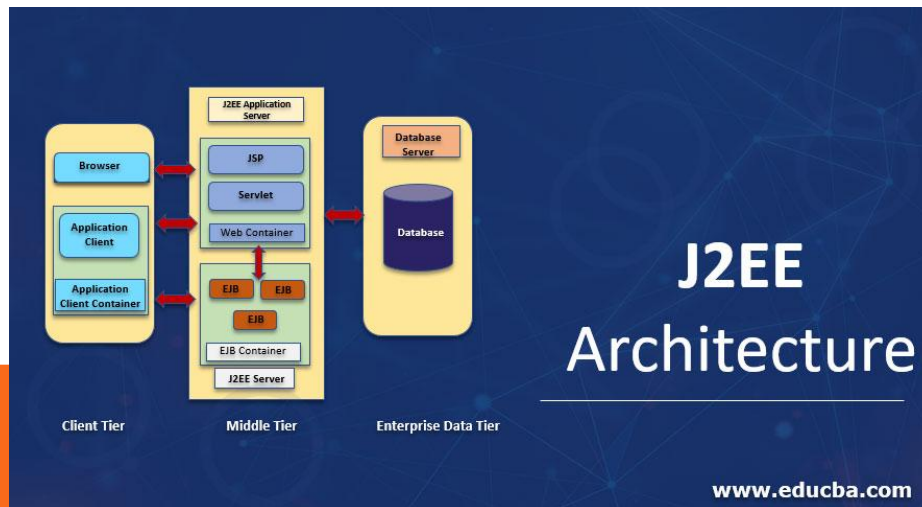
Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Null, Strategy, Template and Visitor Come under Behavioral Pattern.



# J2EE PATTERNS

These Patterns are specifically concerned with presentation tier

It was identified by Sun Java Center at Menlo Park in California USA.



# ARCHITECTURAL DESIGN

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.

The Various Designs are

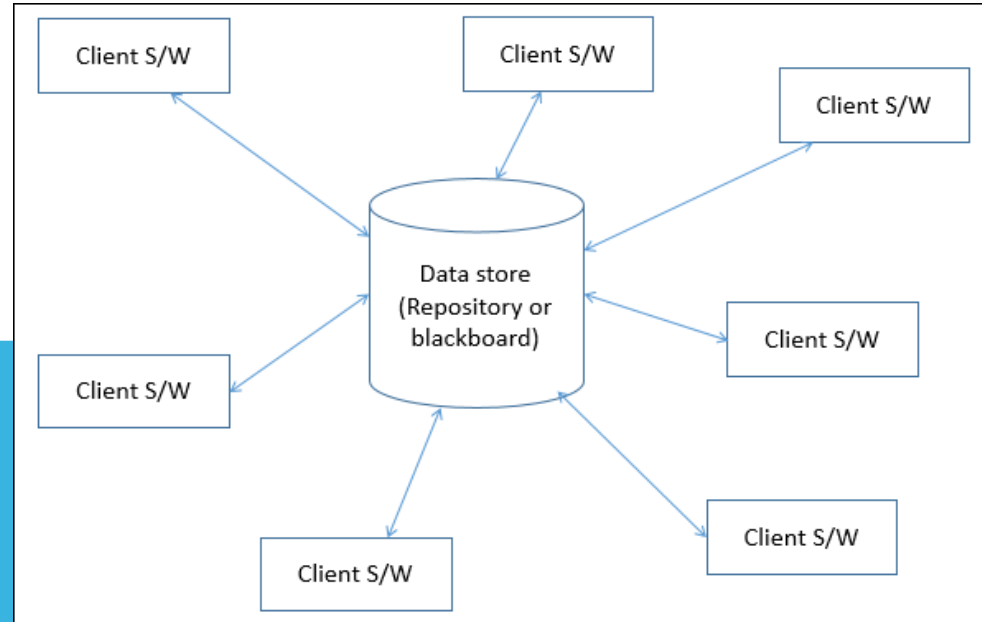
- Data centered
- Data Flow
- Call and Return
- Object Oriented
- Layered



# DATA CENTERED

A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.

This data-centered architecture will promote integrity

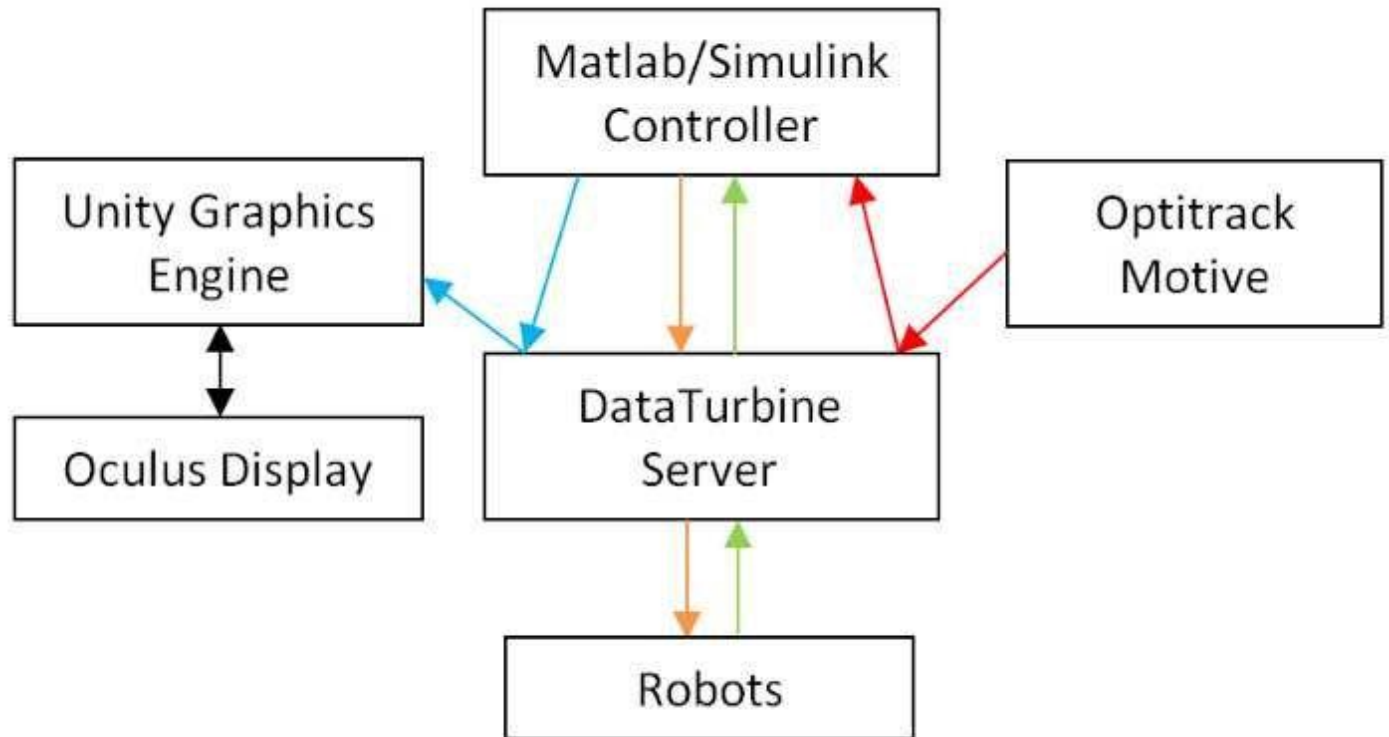


# DATA FLOW ARCHITECTURE

- This kind of architecture is used when input data to be transformed into output data through a series of computational manipulative components.
- Pipes are used to transmit data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don't require any knowledge of the working of neighboring filters.



## DATA FLOW ARCHITECTURE



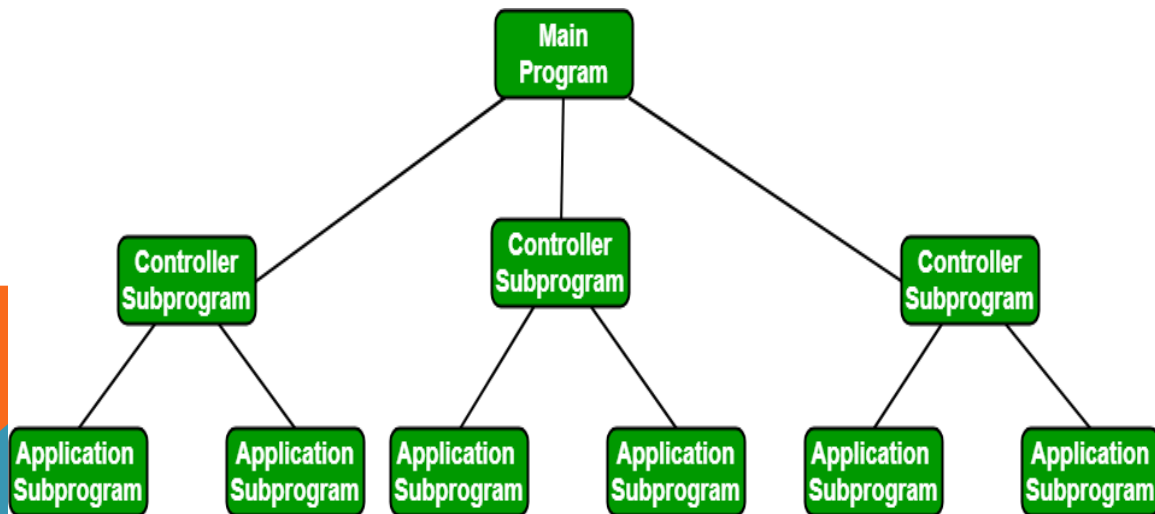
## DataTurbine Channels

- Robot position data
- Grayscale sensor data
- Robot velocity commands
- Position/sensor data

# CALL AND RETURN ARCHITECTURES

It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category

Sub Categories Include: Remote Procedure call and Main program-Sub program Architectures.



# OBJECT ORIENTED ARCHITECTURE

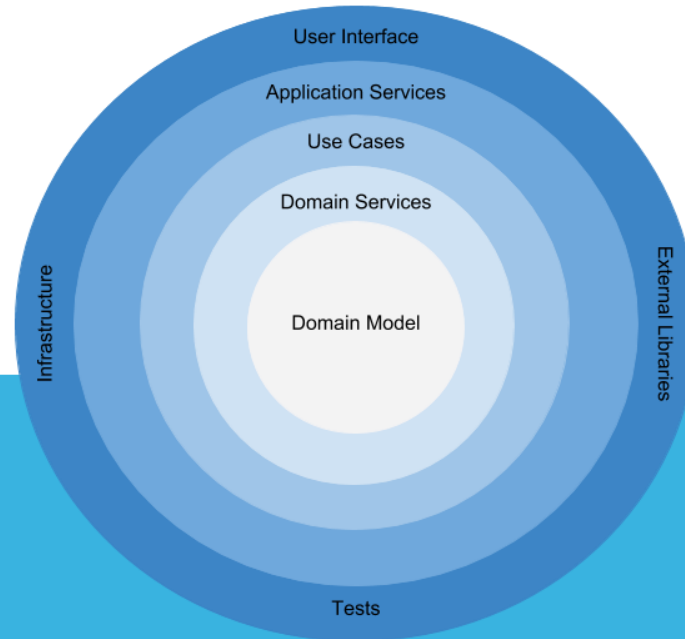
- The components of a system encapsulate data and the operations that must be applied to manipulate the data.
- The coordination and communication between the components are established via the message passing.



Fig. Advantages of Object Oriented Architecture

# LAYERED ARCHITECTURE

- A number of different layers are defined with each layer performing a well-defined set of operations.
- Each layer will do some operations that becomes closer to machine instruction set progressively.



# CONCEPTUAL MODEL

- It is a representation of a system that uses concepts and ideas to form said representation.  
Used across many fields ranging from sciences to socioeconomics to software development.
- These Models try to capture people's understanding of what is being modeled.



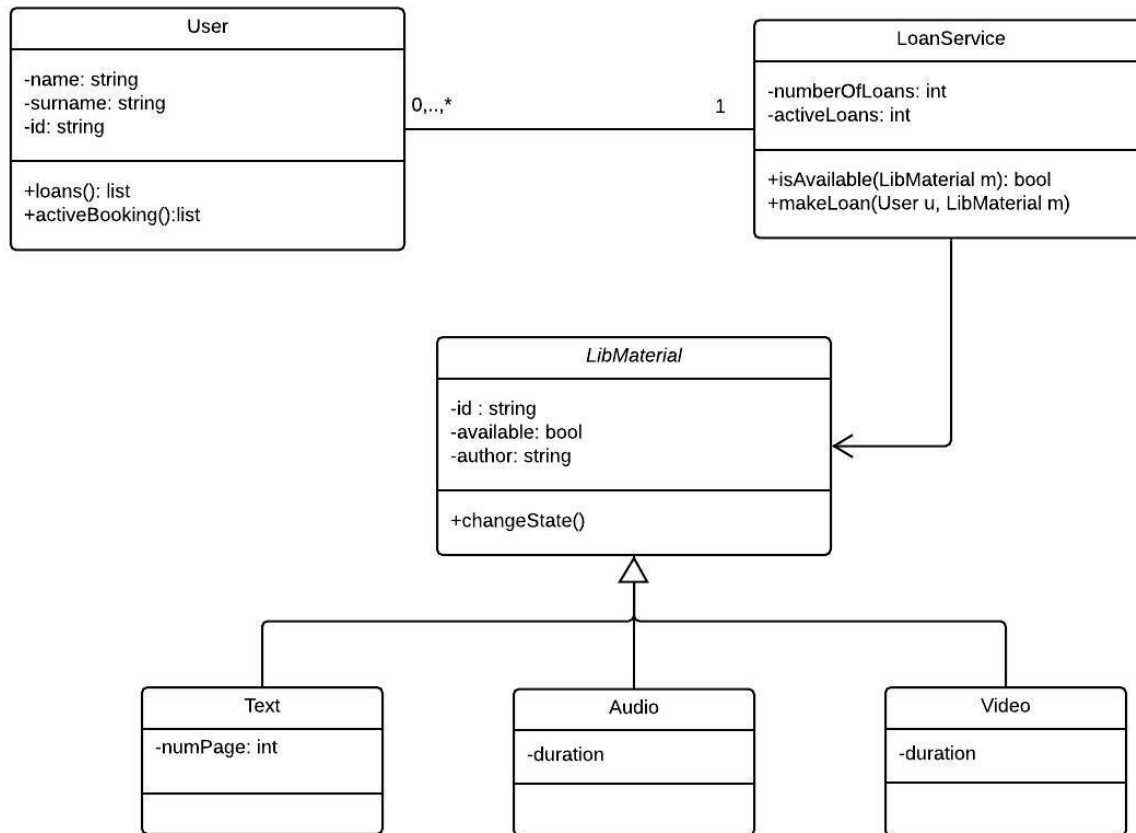
# CLASS DIAGRAM

It is a Static diagram representing the static view of an application. It is not only used for visualizing , describing and documenting aspects but also for constructing executable code.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.



# CLASS DIAGRAMS



# SEQUENCE DIAGRAMS-UML

UML is a modelling Language in the field of software engineering which aims to set standard ways to visualize the design of a system.

The Sequence Diagram is an Interaction Diagram.

It Simply depicts interaction between objects in a sequential order.



# SEQUENCE DIAGRAMS

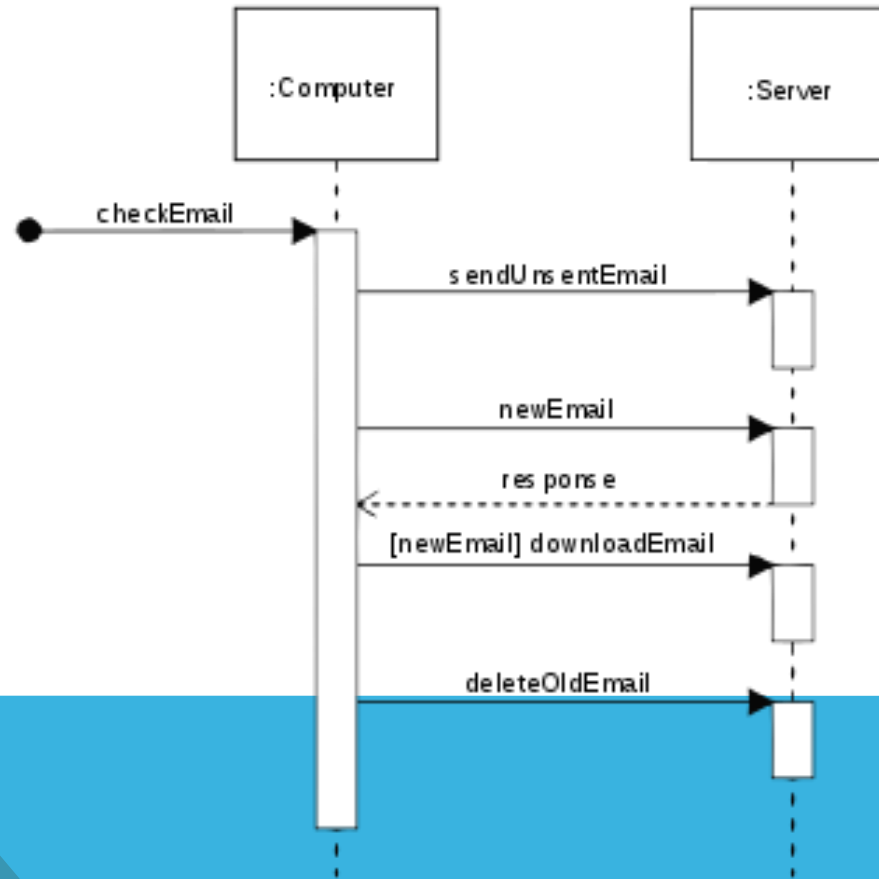
We also use the terms event diagrams or event scenarios to refer to the diagram.

Describe how and in what order the objects in a system function.

The Parts are : Actors, Lifelines, Messages, Guards,



# SEQUENCE DIAGRAM



# ADVANTAGE OF SEQUENCE DIAGRAM

Used to model and visualize the logic behind a sophisticated function, operation or procedure.

They are also used to show details of UML use case diagrams.

Used to understand the detailed functionality of current or future systems.

Visualize how messages and tasks move between objects or components in a system



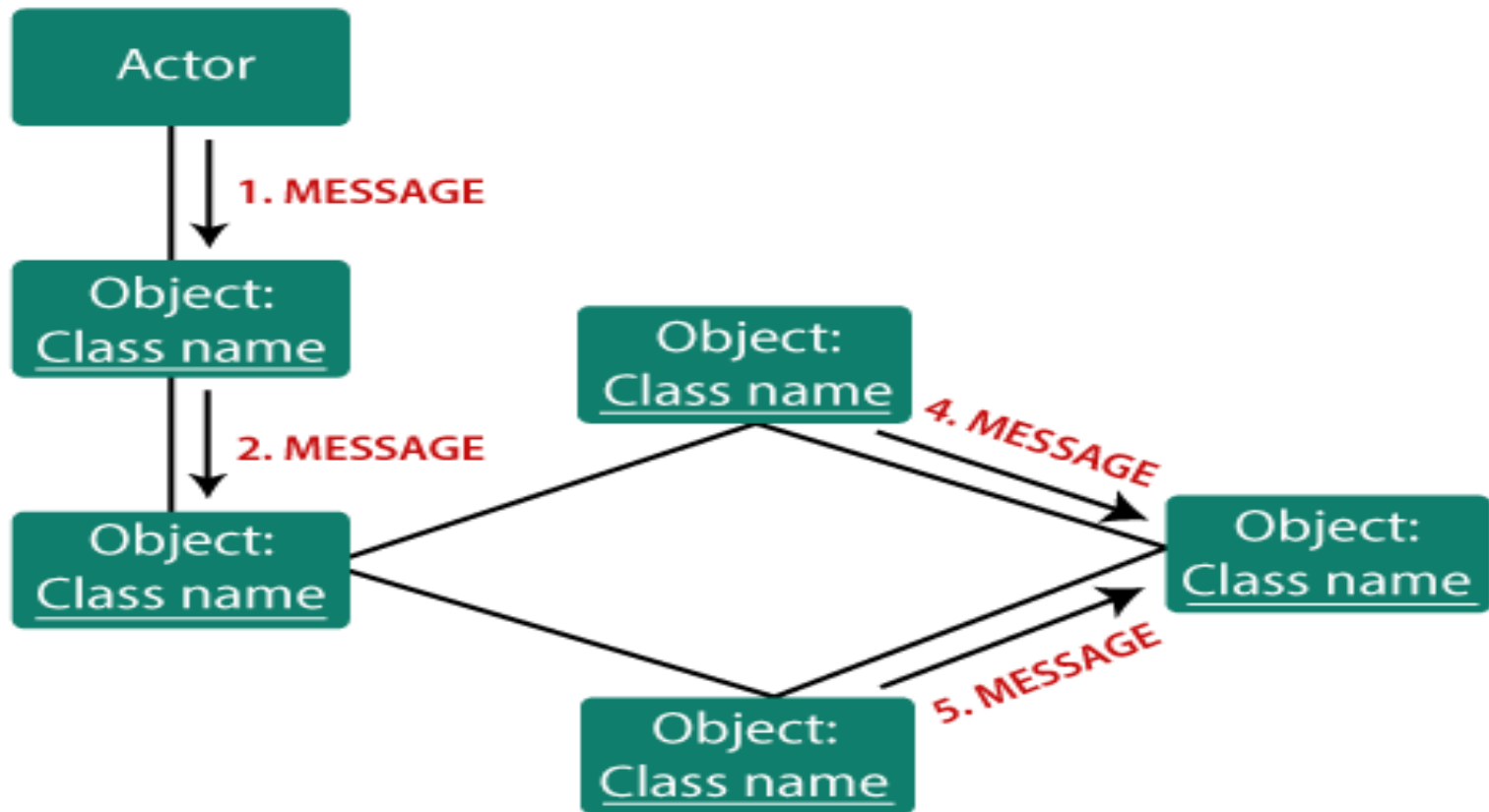
# COLLABORATION DIAGRAMS

- A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). These diagrams can be used to portray the dynamic behavior of a particular use case and define the role of each object.




# COLLABORATION DIAGRAM

## Components of a collaboration diagram



# ADVANTAGES OF COLLABORATION DIAGRAM

- The collaboration diagram is also known as Communication Diagram.
  - It mainly puts emphasis on the structural aspect of an interaction diagram, i.e., how lifelines are connected.
  - The syntax of a collaboration diagram is similar to the sequence diagram; just the difference is that the lifeline does not consist of tails.
  - The special case of a collaboration diagram is the object diagram.
  - It focuses on the elements and not the message flow, like sequence diagrams.
- 

# DISADVANTAGE OF COLLABORATION DIAGRAM

Multiple objects residing in the system can make a complex collaboration diagram, as it becomes quite hard to explore the objects.


It is a time-consuming diagram.

After the program terminates, the object is destroyed.

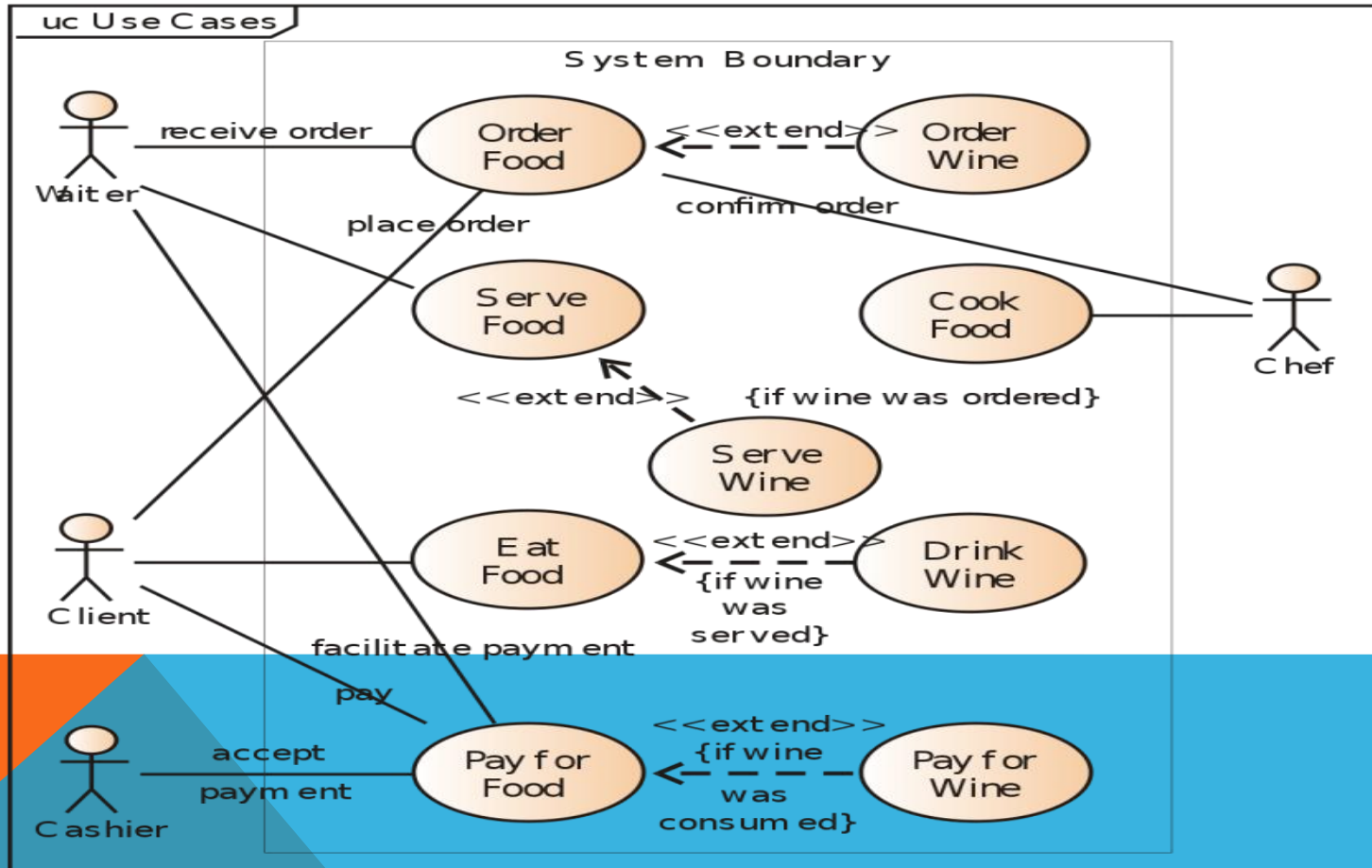
As the object state changes momentarily, it becomes difficult to keep an eye on every single that has occurred inside the object of a system.



# UML USE CASE DIAGRAM

- A use case diagram is used to represent the dynamic behavior of a system.
  - It encapsulates the system's functionality by incorporating use cases, actors, and their relationships.
  - It models the tasks, services, and functions required by a system/subsystem of an application.
  - It depicts the high-level functionality of a system and also tells how the user handles a system.
  - Accumulates system's requirement.
- 
- A decorative graphic at the bottom of the slide consisting of three overlapping triangles. The leftmost triangle is orange, the middle one is a darker blue, and the rightmost one is a lighter blue.

# UML USE CASE DIAGRAM



# UML USE CASE DIAGRAM BENEFITS

It gathers the system's needs.

It depicts the external view of the system.

It recognizes the internal as well as external factors that influence the system.

It represents the interaction between the actors.



# COMPONENT DIAGRAM

Component diagrams are different in terms of nature and behavior.

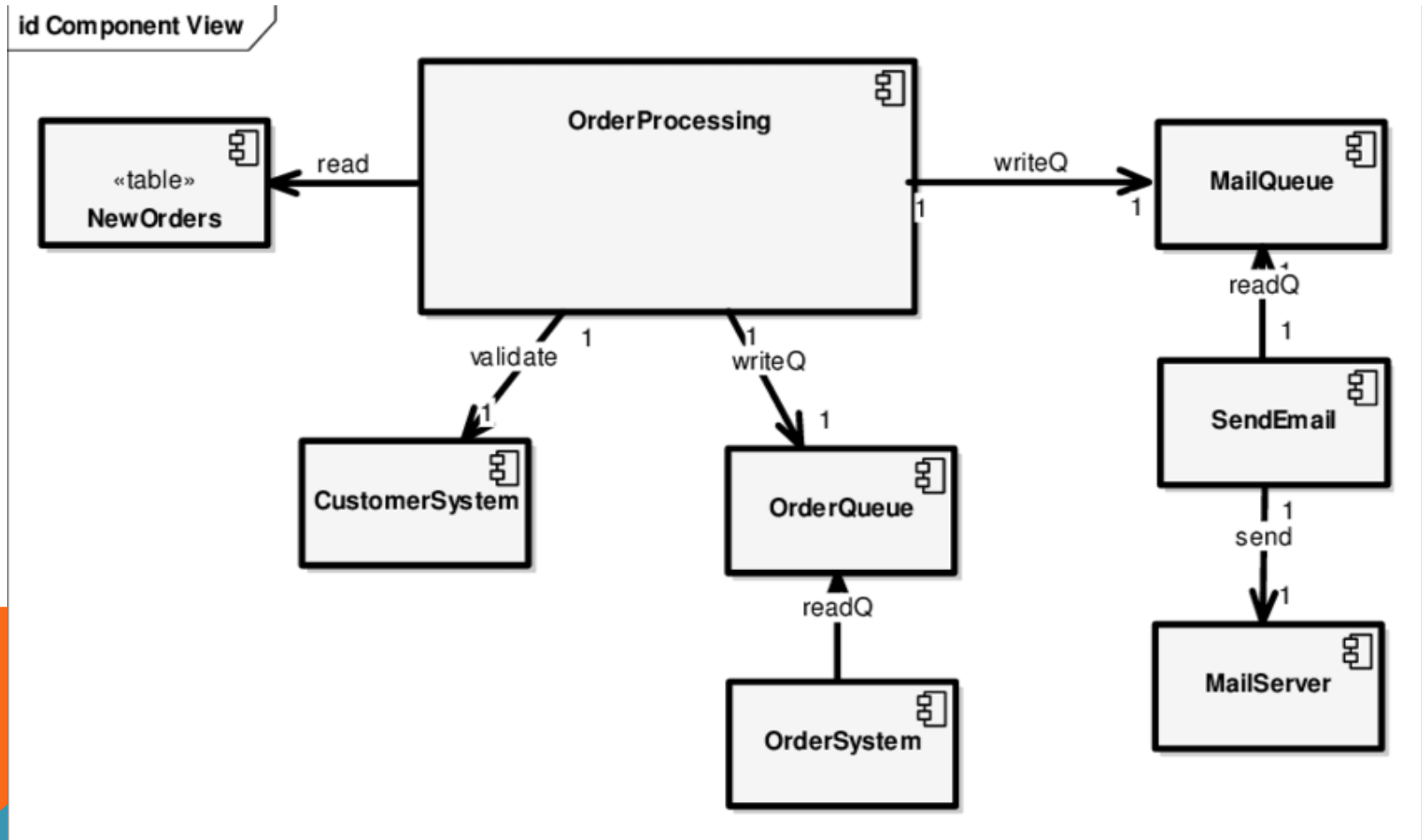
Component diagrams are used to model the physical aspects of a system.

Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.

Component diagrams are used to visualize the organization and relationships among components in a system.



# COMPONENT DIAGRAM



# UML COMPONENT DIAGRAM

## ADVANTAGES

Component diagrams are very simple, standardized, and very easy to understand.

It is also useful in representing implementation of system.

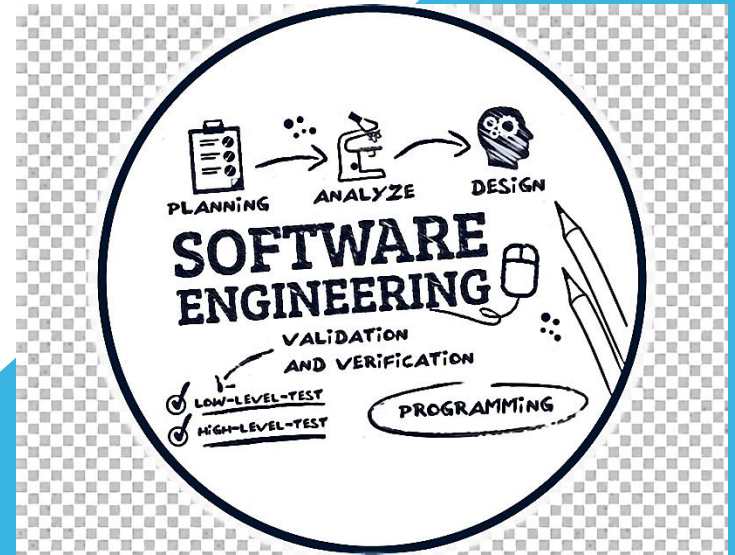
These are very useful when you want to make a design of some device that contains an input-output socket.

Use of reusable components also helps in reducing overall development cost.

It is very easy to modify and update implementation without any causing any other side effects.



## CHAPTER 4



# CONTENTS PART-I

A strategic approach to software testing

Testing Strategies

Black Box and White Box Testing

Validation Testing

System Testing

Art of Debugging



# SOFTWARE TESTING

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free.

It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.



# A STRATEGIC APPROACH TO TESTING

According to Glenn Myers, The objectives are :

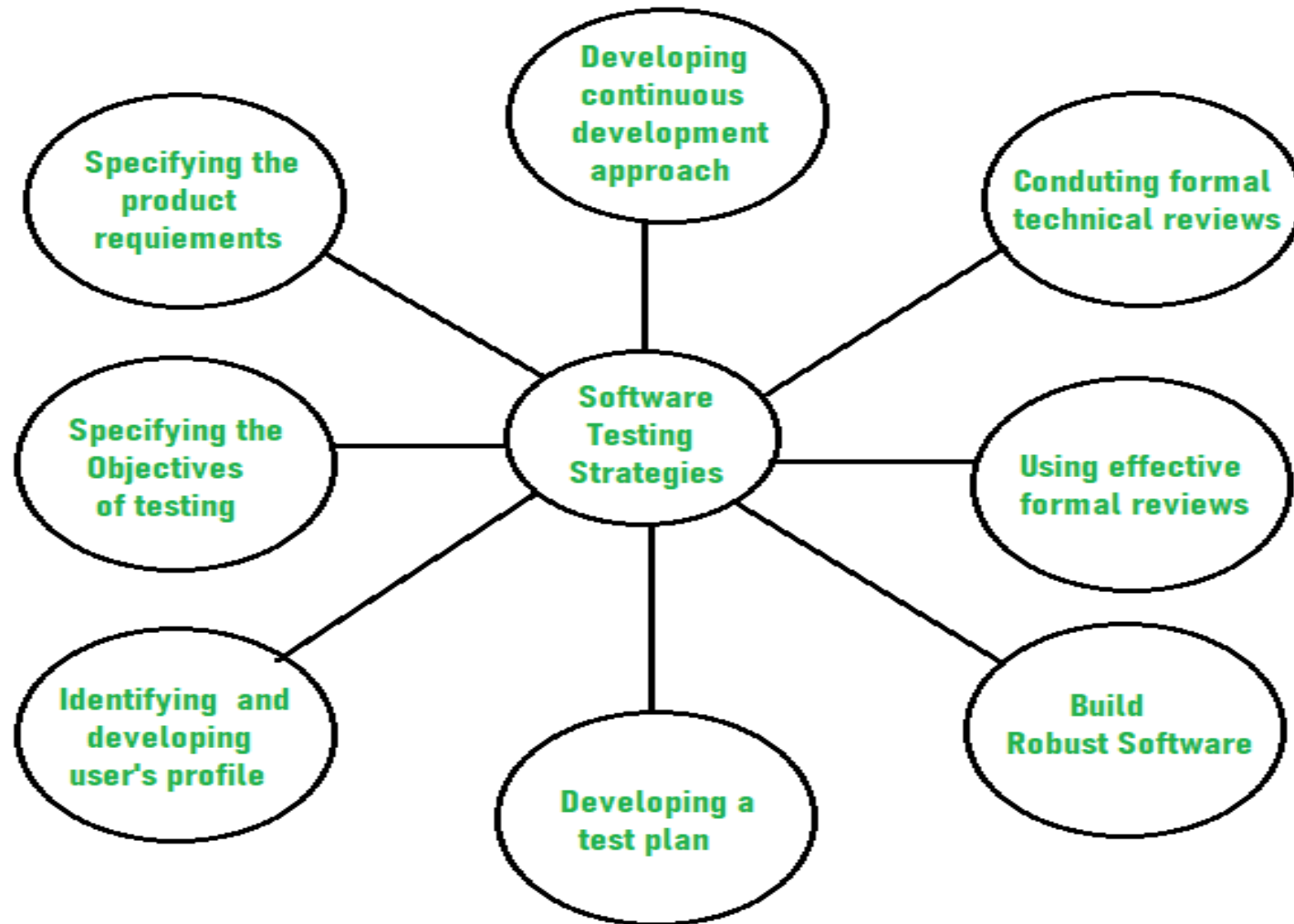
The process of investigating and checking a program to find whether there is an error or not and does it fulfill the requirements or not is called testing.

When the number of errors found during the testing is high, it indicates that the testing was good and is a sign of good test case.

Finding an unknown error that's wasn't discovered yet is a sign of a successful and a good test case.



# VARIOUS TESTING STRATEGIES



# CHARACTERISTICS OF TESTING

The developer should conduct the successful technical reviews.

Testing starts with the component level and work from outside toward the integration.

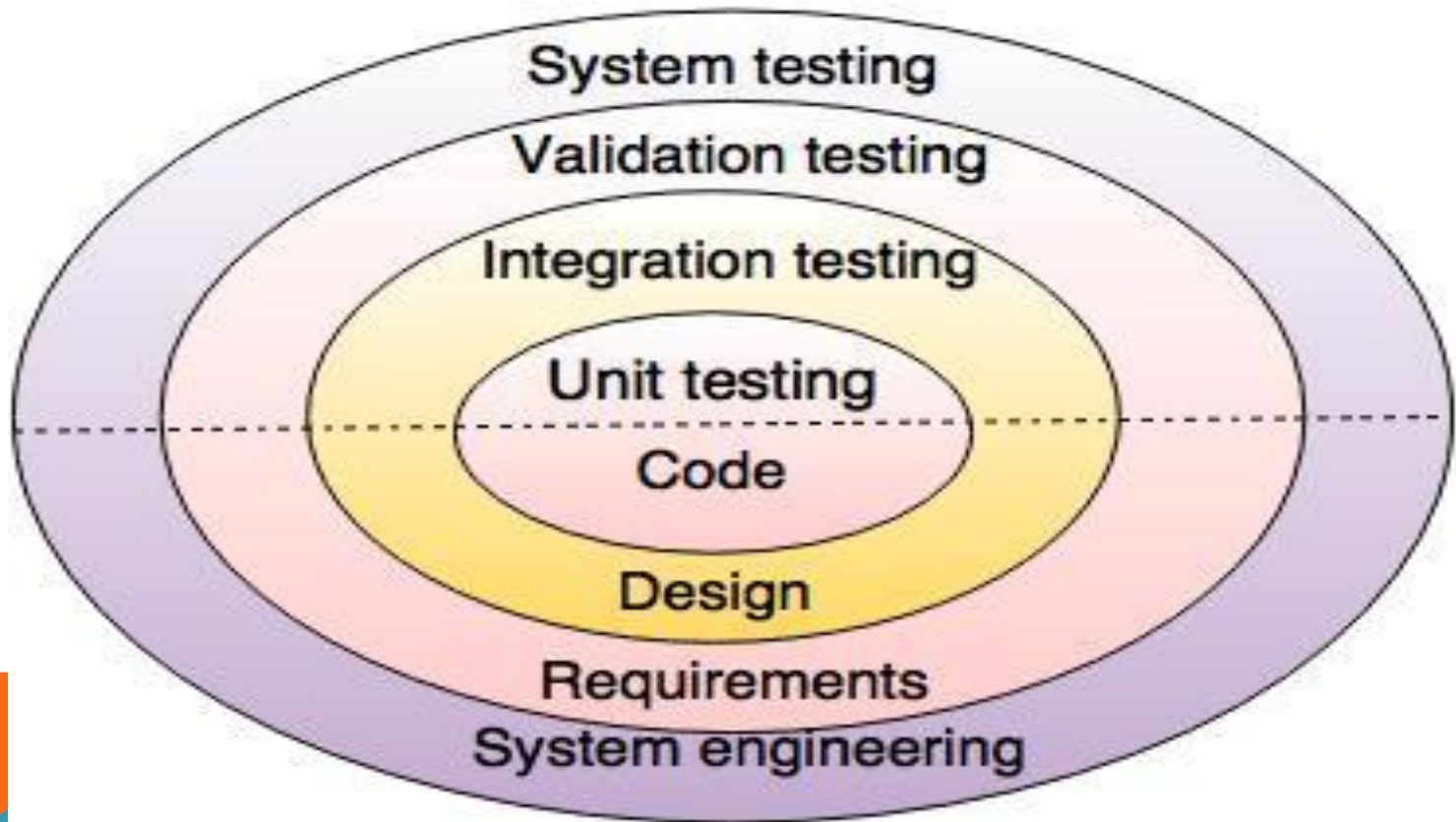
Different testing techniques are suitable at different point in time.

Testing is organized by the developer of the software and by an independent test group.

Debugging and testing are different activities, then also the debugging should be accommodated in any strategy of testing.



# SPIRAL TESTING STRATEGY

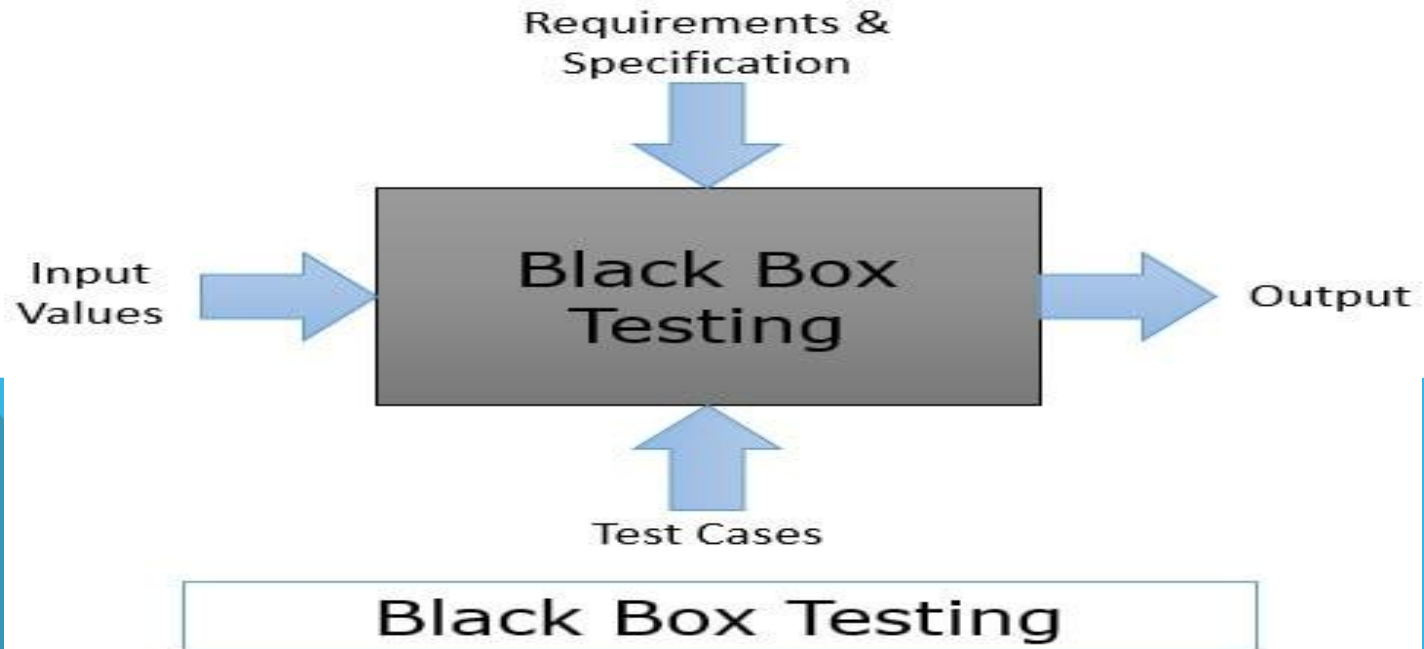


**Fig. - Testing Strategy**

# BLACK BOX TESTING

A type of Software Testing where the functionality of software is not known.

The Testing is done without internal knowledge of products.



# SYNTAX DRIVEN TESTING

1. Syntax Driven : This type of testing is applied to systems that can be syntactically represented by some language.

Example: Compilers, Language that can be represented by context free grammar

Each grammar rule is used once.



# EQUIVALENCE PARTITIONING

The idea is to partition the input domain of the system into a number of equivalence classes such that each member of class works in a similar way, i.e.,

If a test case in one class results in some error, other members of class would also result into same error.

Has Two Steps: Identification of equivalence classes and generating test cases. For each class.



# BOUNDARY VALUE ANALYSIS

Boundaries are very good places for errors to occur. Hence if test cases are designed for boundary values of input domain then the efficiency of testing improves and probability of finding errors also increase.

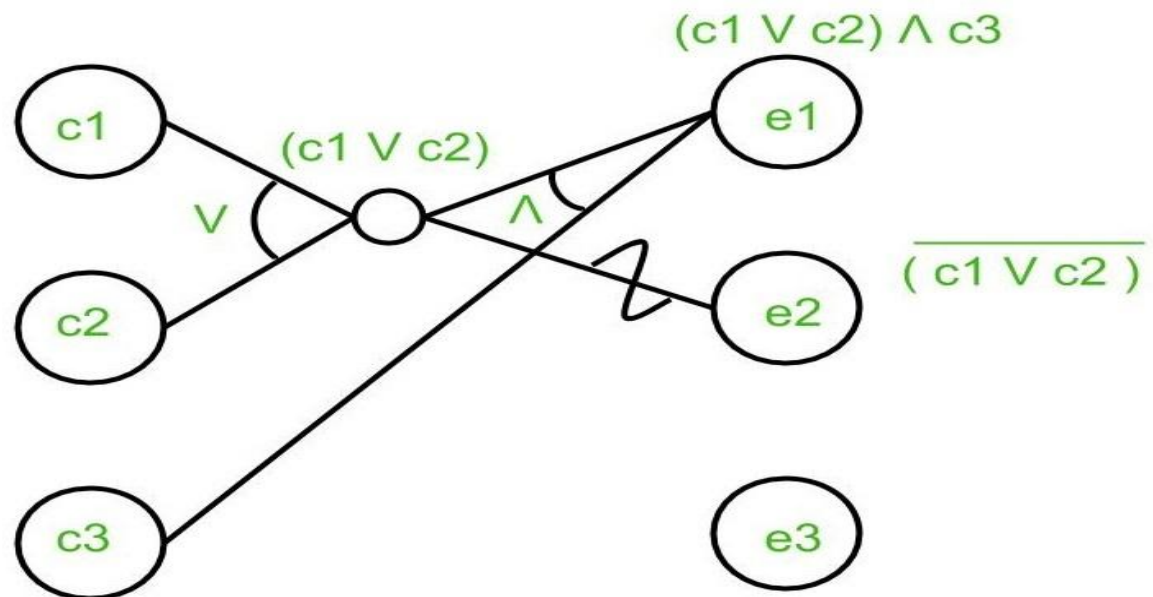
For example – If valid range is 10 to 100 then test for 10,100 also apart from valid and invalid inputs.



# CAUSE EFFECT GRAPHING

This technique establishes relationship between logical input called causes with corresponding actions called effect.

The causes and effects are represented using Boolean graphs.



# REQUIREMENT BASED TESTING

This Testing Includes Validating the requirements given in the Software Requirement Specification Document for a particular software system.

Testing must be carried out in a timely manner.

Testing process should add value to the software life cycle, hence it needs to be effective.

Testing must provide the overall status of the project, hence it should be manageable.



# COMPATIBILITY TESTING

The test case result not only depend on product but also infrastructure for delivering functionality. When the infrastructure parameters are changed it is still expected to work properly.

Some of them include processor, architecture, back-end components and OS.



# WHITE BOX TESTING

White box testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing.

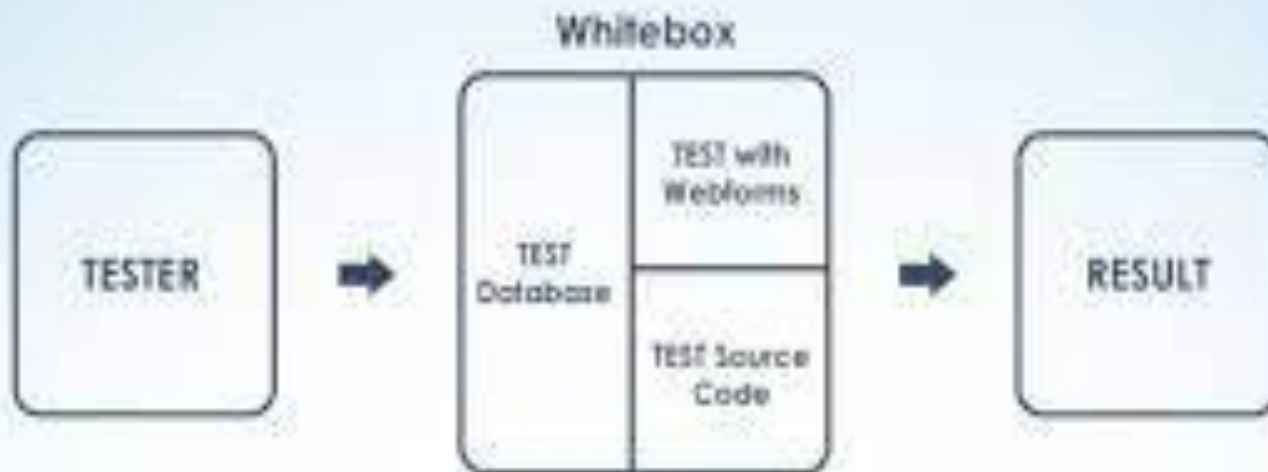
It is also called glass box testing or clear box testing or structural testing.

The Process Steps Include: Input, Processing , Proper Test Planning and Output.



# WHITE BOX TESTING

## WHITE BOX TESTING APPROACH



# WHITE BOX TESTING TECHNIQUES

**Statement Coverage:** In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested. In case of a flowchart, every node must be traversed at least once

**Branch Coverage** In this technique, test cases are designed so that each branch from all decision points are traversed at least once.

**Condition Coverage:** In this technique, all individual conditions must be covered.



# WHITE BOX TESTING TECHNIQUES

**Multiple Condition Coverage:** In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once.

**Basis Path Testing:** In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

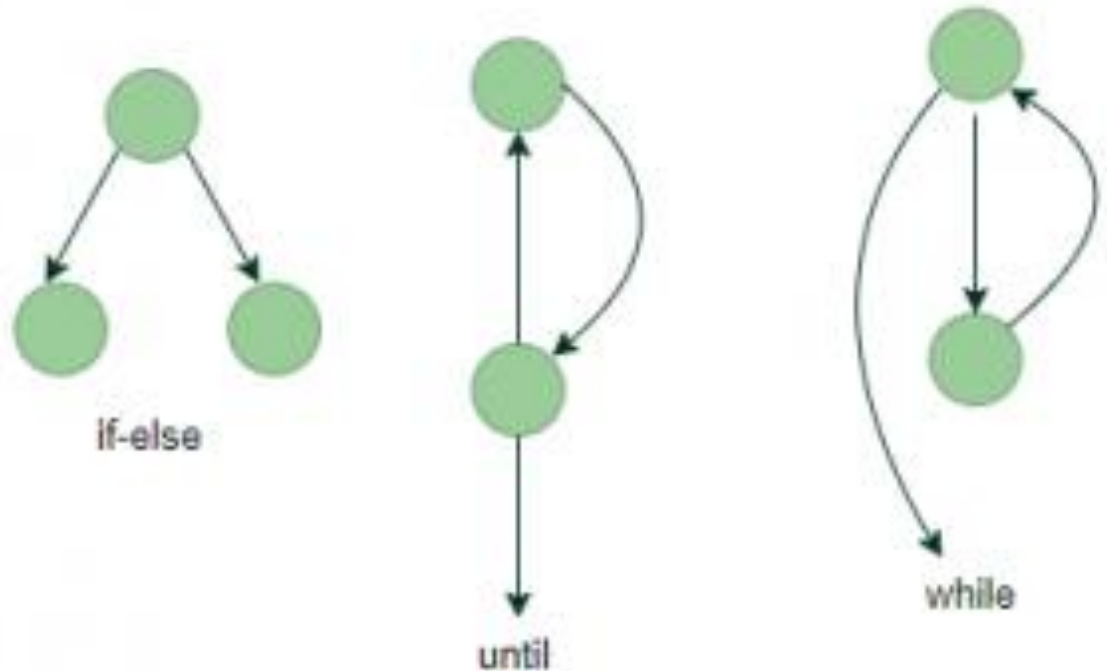


# WHITE BOX TESTING TECHNIQUES

Flow graph notation: It is a directed graph consisting of nodes and edges. Each node represents a sequence of statements, or a decision point.

A predicate node is the one that represents a decision point that contains a condition after which the graph splits.

Regions are bounded by nodes and edges.



# ADVANTAGES OF WHITE BOX TESTING

White box testing is very thorough as the entire code and structures are tested.

It results in the optimization of code removing error and helps in removing extra lines of code.

It can start at an earlier stage as it doesn't require any interface as in case of black box testing.

Easy to automate.



# DISADVANTAGES OF WHITE BOX TESTING

Main disadvantage is that it is very expensive.

Redesign of code and rewriting code needs test cases to be written again.

Testers are required to have in-depth knowledge of the code and programming language as opposed to black box testing.

Missing functionalities cannot be detected as the code that exists is tested.

Very complex and at times not realistic.



# DIFFERENCES BETWEEN WHITE BOX AND BLACK BOX

Black Box Testing	White Box Testing
1. Black box testing techniques are also called functional testing techniques.	1. White box testing techniques are also called structural testing techniques.
2. Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	2. White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
3. It is mainly applicable to higher levels of testing such as Acceptance Testing and System Testing	3. Mainly applicable to lower levels of testing such as Unit Testing and Integration Testing
4. Black box testing is generally done by Software Testers	4. White box testing is generally done by Software Developers
5. Programming knowledge is not required	5. Programming knowledge is required
6. Implementation knowledge is not required.	6. Implementation knowledge is required

# VALIDATION TESTING

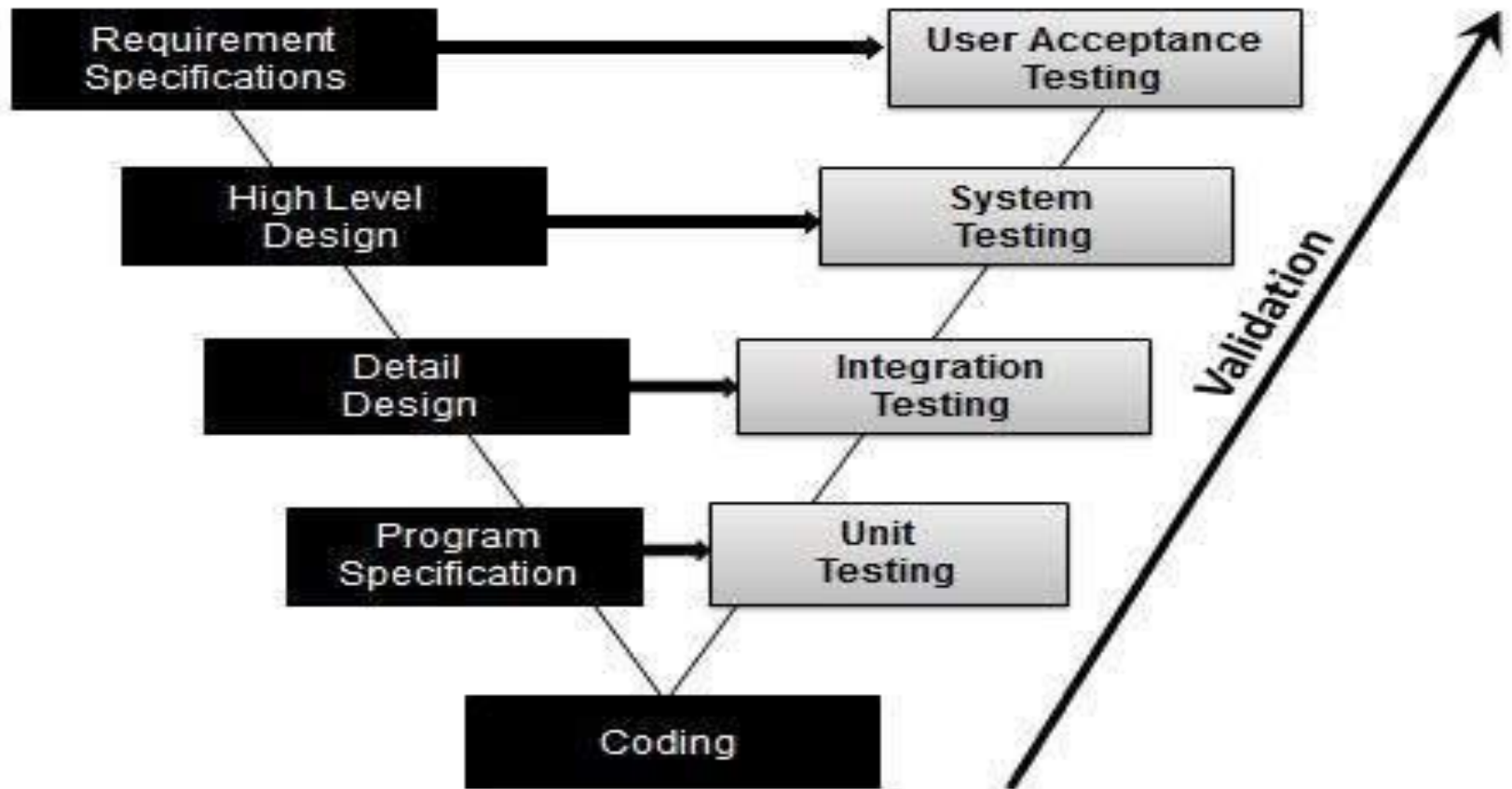
The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

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

Validation Testing ensures that the product actually meets the client's needs.



# VALIDATION TESTING V-MODEL



# VALIDATION TESTING TYPES

Unit Testing

Integration Testing

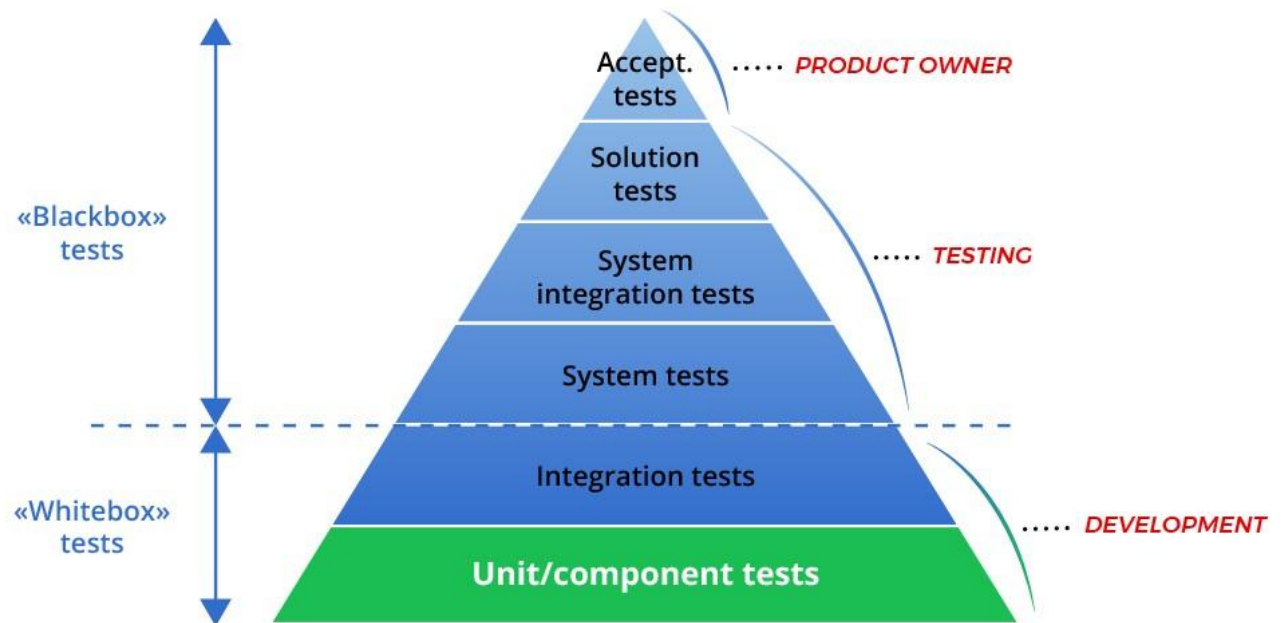
System Testing

User Accepting Testing



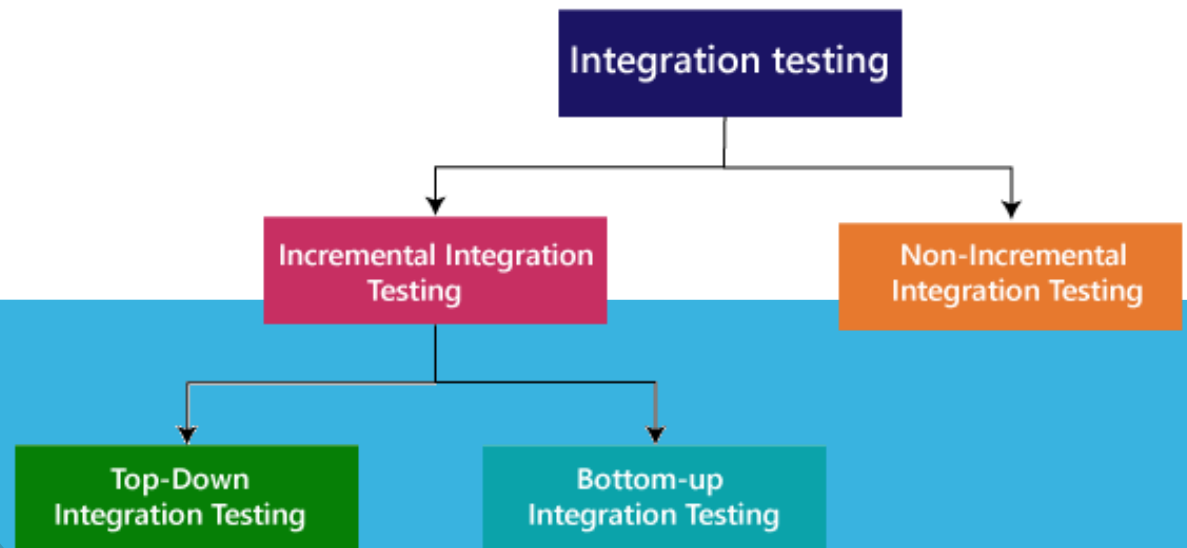
# UNIT TESTING

It is an important type of validation testing. The point of the unit testing is to search for bugs in the product segment. Simultaneously, it additionally confirms crafted by modules and articles which can be tried independently.



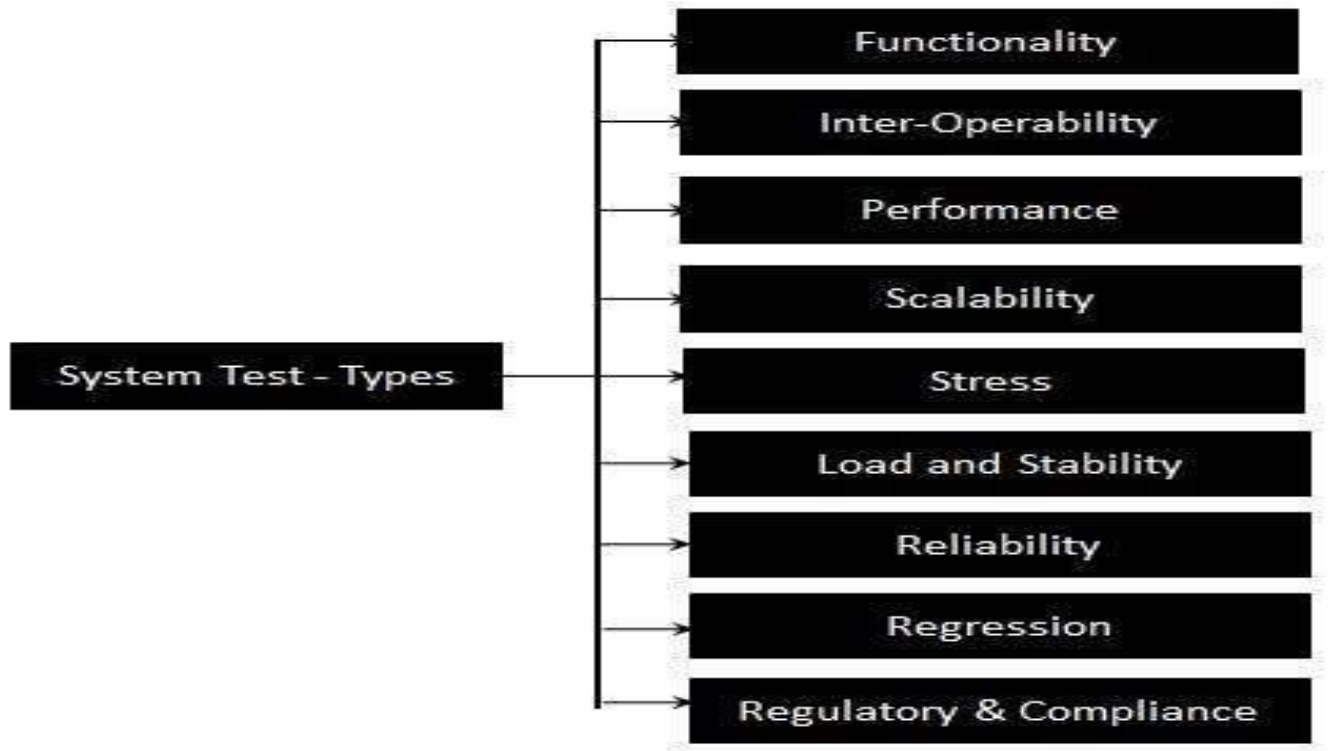
# INTEGRATION TESTING

This is a significant piece of the validation model wherein the interaction between, where the association between the various interfaces of the pertaining component is tried.



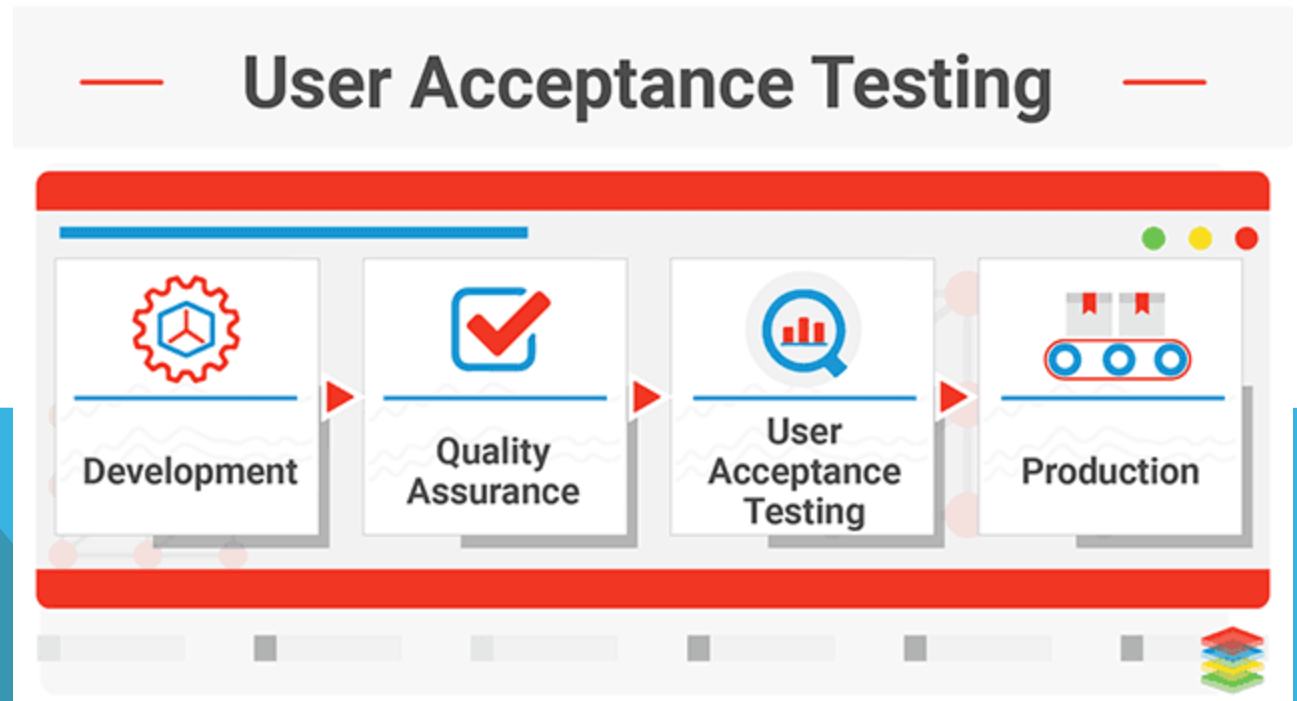
# SYSTEM TESTING

System testing is done when the whole programming framework is prepared. The principal worry of framework testing is to confirm the framework against the predefined necessities.



# USER ACCEPTANCE TESTING

During this testing, the tester actually needs to think like the customer and test the product concerning client needs, prerequisites, business forms and decide if the product can be given over to the customer or not.



# DEBUGGING

In the context of software engineering, debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing and removing errors.

This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software

- . It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.



# DEBUGGING PROCESS

Problem identification and report preparation.

Assigning the report to software engineer to the defect to verify that it is genuine.

Defect Analysis using modeling, documentations, finding and testing candidate flaws, etc.

Defect Resolution by making required changes to the system.

Validation of corrections.



# DEBUGGING TOOLS

Debugging tool is a computer program that is used to test and debug other programs. A lot of public domain software like gdb and dbx are available for debugging.

Some of the widely used debuggers are:

Radare2

Win Dbg

Valgrind



# DEBUGGING VS TESTING

Impact QA QUALITY REDEFINED			Differentiation Between Testing and Debugging	
	Testing		Debugging	
1	Testing is done by the tester		Debugging is done by either programmer or developer	
2	There is no need for design knowledge in the testing process		Debugging can't be done without proper design knowledge	
3	Testing can be manual or automated		Debugging is always manual. Debugging can't be automated.	
4	Testing is initiated after the code is written		Debugging commences with the execution of a test case	
5	Testing is a stage of the software development life cycle (SDLC)		Debugging is not an aspect of software development life cycle, it occurs as a consequence of testing	

# **SOFTWARE ENGINEERING**

PART-II

# CONTENTS

Software Quality

Metrics for Analysis Model

Metrics for Design Model

Metrics for Source Code

Metrics for Testing

Metrics for Maintenance.



# SOFTWARE QUALITY

Software quality is defined as a field of study and practice that describes the desirable attributes of software products. There are two main approaches to software quality: defect management and quality attributes.

Software Quality refers to both functional quality and structural quality.



# SOFTWARE FUNCTIONAL QUALITY

It reflects how well it satisfies a given design, based on the functional requirements or specifications.

SFQ is pertaining to conformance to the functional requirements.

The SFQ is measured by the level of end user satisfaction.



# SOFTWARE STRUCTURAL QUALITY

It deals with the handling of non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability, and the degree to which the software was produced correctly

Attributes are : Code testability, Maintainability, understandability, efficiency , security.



# SOFTWARE QUALITY ASSURANCE

Is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly.

Software Quality Assurance is a process which works parallel to development of a software

Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process.



# SOFTWARE QUALITY CONTROL

Software Quality Control (SQC) is a set of activities to ensure the quality in software products.

These activities focus on determining the defects in the actual products produced.

It involves product-focused action.

Software Quality Control is commonly referred to as Testing.



# SOFTWARE QUALITY CHALLENGE

In the software industry, the developers will never declare that the software is free of defects, unlike other industrial product manufacturers usually do.

The Key Reasons are :

- Product Complexity
- Product Visibility
- Product Development and Production Process.



# METRICS FOR ANALYSIS MODEL

Technical work in software engineering begins with the creation of the analysis model. It is at this stage that requirements are derived and that a foundation for design is established.

Therefore, technical metrics that provide insight into the quality of the analysis model are desirable.

These Metrics are used to analyze the analysis model with the objective of increased coding, integration and testing effort.

Ex: Function Point(FP ) and Lines of Code(LOC)



# METRICS FOR DESIGN MODEL

The success of a software project depends largely on the quality and effectiveness of the software design.

Hence, it is important to develop software metrics from which meaningful indicators can be derived.

Various design metrics such as architectural design metrics, component-level design metrics, user-interface design metrics, and metrics for object-oriented design are used to indicate the complexity, quality, and so on of the software design.



# METRICS FOR SOURCE CODE

Halstead proposed the first analytic laws for Computer science by using a set of primitive measures, which can be derived once the design phase is complete and code is generated. These measures are listed below.

$n_1$  = number of distinct operators in a program

$n_2$  = number of distinct operands in a program

$N_1$  = total number of operators

$N_2$  = total number of operands.

The Halstead Equation denotes the Coding metric for software quality.

$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$ . [Program Length]

$V = N \log_2 (n_1 + n_2)$ . [Program Volume]

# METRICS FOR TESTING

Majority of the metrics used for testing focus on testing process rather than the technical characteristics of test. Generally, testers use metrics for analysis, design, and coding to guide them in design and execution of test cases.

Halstead measures can be used to derive metrics for testing effort. By using program volume (V) and program level (PL), Halstead effort (e) can be calculated by the following equations.

$$e = V / PL$$

$$\text{Percentage of testing effort (z)} = e(z) / \sum e(i)$$



# METRICS FOR MAINTENANCE

For the maintenance activities, metrics have been designed explicitly. IEEE have proposed Software Maturity Index (SMI), which provides indications relating to the stability of software product. For calculating SMI, following parameters are considered.

Number of modules in current release ( $M_T$ )

Number of modules that have been changed in the current release ( $F_e$ )

Number of modules that have been added in the current release ( $F_a$ )

Number of modules that have been deleted from the current release ( $F_d$ )

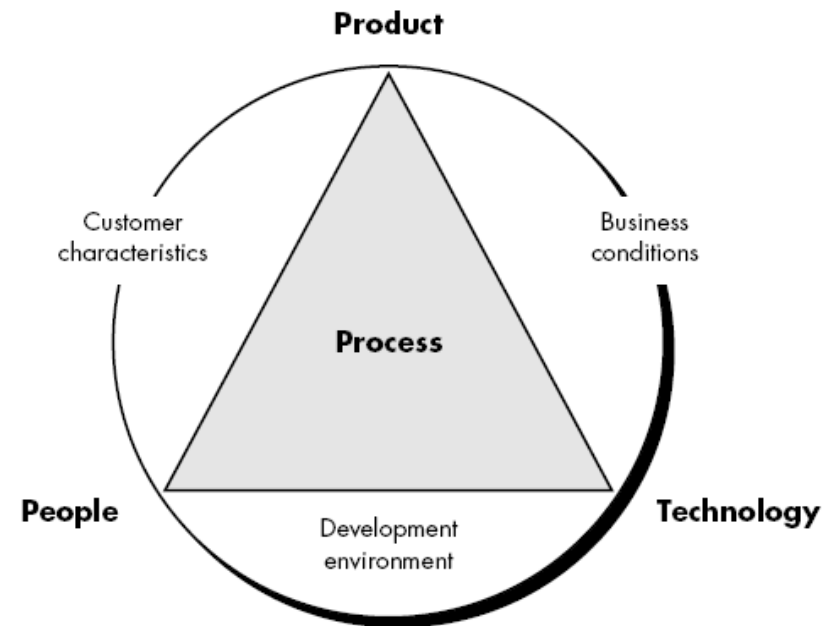
Once all the parameters are known, SMI can be calculated by using the following equation.

$$SMI = [M_T - (F_a + F_e + F_d)] / M_T$$

# PROCESS METRICS

To improve any process, it is necessary to measure its specified attributes, develop a set of meaningful metrics based on these attributes, and then use these metrics to obtain indicators in order to derive a strategy for process improvement.

Using software process metrics, software engineers are able to assess the efficiency of the software process that is performed using the process as a framework



# PRODUCT METRICS

Product metrics are software product measures at any stage of their development, from requirements to established systems. Product metrics are related to software features only.

Metrics are of 2 types :

- Dynamic metrics that are collected by measurements made from a program in execution.
- Static metrics that are collected by measurements made from system representations such as design, programs, or documentation.



# DYNAMIC PRODUCT METRICS

Dynamic metrics are usually quite closely related to software quality attributes. It is relatively easy to measure the execution time required for particular tasks and to estimate the time required to start the system. These are directly related to the efficiency of the system failures and the type of failure can be logged and directly related to the reliability of the software.



# STATIC PRODUCT METRICS

Static metrics have an indirect relationship with quality attributes. A large number of these matrices have been proposed to try to derive and validate the relationship between the complexity, understandability, and maintainability.



## CHAPTER 5

# CONTENTS

Software Measurement

Metrics for Software quality

Reactive vs Proactive Risk Management

Software Risks

Risk Identification

Risk Projection

Risk Refinement

RMMM

RMMM Plan



# CONTENTS

Quality Concepts

Software Quality assurance

Software Reviews

Formal Technical Reviews


Statistical Software Quality Assurance

Software Reliability

ISO 9000 Quality Standards



# SOFTWARE MEASUREMENT

- Software Measurement: A measurement is an manifestation of the size, quantity, amount or dimension of a particular attributes of a product or process.
  - Software measurement is a titrate impute of a characteristic of a software product or the software process.
  - It is an authority within software engineering. Software measurement process is defined and governed by ISO Standard.
  - Software Measurement is used as a parameter for the manifestation of software.
- 

# NEEDS OF SOFTWARE MEASUREMENT

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

These are the 4 basic needs of software measurement.



# TYPES OF SOFTWARE MEASUREMENT

- There are 2 types of Software Measurement

→ Direct Measurement: In direct measurement the product, process or thing is measured directly using standard scale.

→ Indirect measurement: In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference.



# METRICS

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

The 4 Metrics are

- Planning
- Organizing
- Controlling
- Improving



# CHARACTERISTICS OF METRICS

Quantitative

Understandable

Applicability

Repeatable

Economical

Language Independent



# CLASSIFICATION OF SOFTWARE METRIC

There are 2 types of Software Metrics

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

**Process Metric:** Process metrics pay particular attention on enhancing the long term process of the team or organization.

**Project Metrics:** Project matrix is describes the project characteristic and execution process.

Number of Developers, Staffing Pattern, Cost and Schedule and Productivity.



# SOFTWARE RISK

- Software risk encompasses the probability of occurrence for uncertain events and their potential for loss within an organization
- Risk management has become an important component of software development as organizations continue to implement more applications across a multiple technology, multi-tiered environment.
- Typically, software risk is viewed as a combination of robustness, performance efficiency, security and transactional risk propagated throughout the system.



# REACTIVE RISK MANAGEMENT

- **Reactive risk management tries to reduce the damage of potential threats and speed an organization's recovery from them, but assumes that those threats will happen eventually.**



# PROACTIVE RISK MANAGEMENT

- As the name suggests, proactive risk management means that you identify risks before they happen and figure out ways to avoid or alleviate the risk. It seeks to reduce the hazard's risk potential or, even better, prevent the threat altogether. A good example here is vulnerability testing and remediation



# REACTIVE VS PROACTIVE

What Type of Company Are You?

## PROACTIVE *companies*

- Are prepared for most eventualities
- Know exactly what steps to take during emergencies
- Recover from disasters faster and easier
- Are better able to satisfy customers during a crisis
- Are less likely to experience employee turnover
- Experience less legal distress

## REACTIVE *companies*

- Are unprepared for sudden events
- Are unsure what to do when emergencies happen
- Recover slowly, if at all
- Are less equipped to continue serving customers
- Are more likely to experience employee turnover
- Are left open to more legal complications

HR   Benefits   Payroll

[gnapartners.com](http://gnapartners.com)

 **G&A Partners**  
Time to grow.

# RISK IDENTIFICATION

- Risk identification is the process of determining risks that could potentially prevent the program, enterprise, or investment from achieving its objectives. It includes documenting and communicating the concern.
- Risk identification is the critical first step of the risk management process

## LESSON SUMMARY



# RISK PROJECTION

- Risk projection, also called risk estimation, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur.
- The project planner, along with other managers and technical staff, performs four risk projection activities:
  - (1) establish a scale that reflects the perceived likelihood of a risk
  - (2) delineate the consequences of the risk,
  - (3) estimate the impact of the risk on the project and the product,
  - (4) note the overall accuracy of the risk projection so that there will be no misunderstandings.



# RISK REFINEMENT

- Process of restating the risks as a set of more detailed risks that will be easier to mitigate, monitor, and manage.
- In this step we actually understand the risk on a much more deeper detail and try to look at a broader perspective on how to handle the risk.
- These Steps are actually the basis for the RMMM Model
- It stands for Risk Mitigation, Monitoring and Management Plan.



# RMMM


- RMMM Stands for Risk Mitigation, Monitoring and Management Plan.
- A risk management technique is usually seen in the software project plan.
- In this plan all works are done as a part of risk analysis
- Risk is documented with the help of a Risk Information Sheet (RIS).
- This RIS is controlled by using a database system for easier management of information i.e creation, priority ordering, searching, and other analysis.
- After documentation of RMMM and start of a project, risk mitigation and monitoring steps will start.

# RISK MITIGATION

- It is an activity used to avoid problems (Risk Avoidance).  
Steps for mitigating the risks as follows.
1. Finding out the risk.
  2. Removing causes that are the reason for risk creation.
  3. Controlling the corresponding documents from time to time.
  4. Conducting timely reviews to speed up the work.



# RISK MONITORING

- It is an activity used for project tracking.  
It has the following primary objectives as follows.
- 1) To Check if Predicted risks occur or not.
  - 2) To Ensure proper application of risk aversion steps defined for risk
  - 3) To Collect data for future risk analysis
  - 4) To allocate what problems are caused by which risks throughout the project.
- 

# RISK MANAGEMENT AND PLANNING

It assumes that the mitigation activity failed and the risk is a reality.

This task is done by Project manager when risk becomes reality and causes severe problems.

If the project manager effectively uses project mitigation to remove risks successfully then it is easier to manage the risks.


This shows that the response that will be taken for each risk by a manager.



# RISK MANAGEMENT LIFECYCLE



# SOFTWARE QUALITY

- Software Quality is defined as a field of study and practice that describes the desirable attributes of software products.
  - Software Quality Product is defined in terms of it's fitness of purpose.
  - Quality Product does Precisely what users want it to do.
  - For software products the fitness of use is generally explained in terms of satisfaction of requirements.
- 

# SOFTWARE QUALITY ASSURANCE

- Software quality assurance (SQA) is a means and practice of monitoring the software engineering processes and methods used in a project to ensure proper quality of the software. It may include ensuring conformance to standards or models, such as ISO/IEC 9126 (now superseded by ISO 25010), SPICE or CMMI.
- It is simply a way to assure quality in the software.
- Set of Activities which ensure processes, procedures as well as Standards



# SOFTWARE REVIEWS

- Software Review is systematic inspection of a software by one or more individuals who work together to find and resolve errors and defects in the software during the early stages of Software Development Life Cycle (SDLC).
- Software review is an essential part of Software Development Life Cycle (SDLC) that helps software engineers in validating the quality, functionality and other vital features and components of the software.
- It is a whole process that includes testing the software product and it makes sure that it meets the requirements stated by the client.



# OBJECTIVE OF SOFTWARE REVIEWS

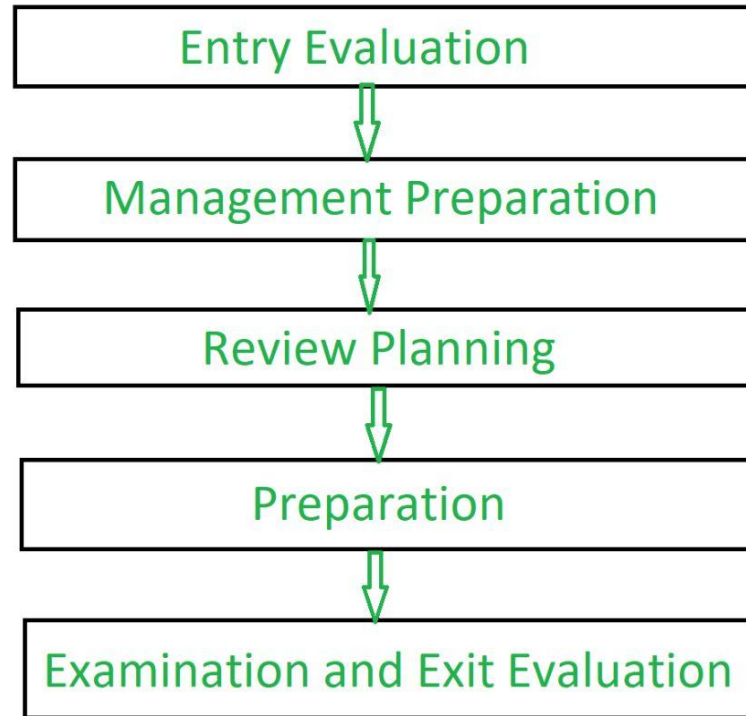
- To Improve the productivity of the development team
- To make the testing process time and cost effective.
- To make the final software and fewer defects
- To Eliminate the inadequacies

To understand the software reviews we need to understand the Process of Software Review.


The Process contains 5 steps that helps us understands the Software review Process during Software Engineering.




# SOFTWARE REVIEW PROCESS



# FORMAL TECHNICAL REVIEW

- Formal Technical Review (FTR) is a software quality control activity performed by software engineers.
  - Useful to uncover error in logic, function and implementation for any representation of the software.
  - The purpose of FTR is to verify that the software meets specified requirements.
  - To ensure that software is represented according to predefined standards.
  - To makes the project more manageable.
  - It helps to review the uniformity in software that is development in a uniform manner.
- 

# STATISTICAL SOFTWARE QUALITY ASSURANCE

- SQA is used to reduce cost and improve the product time to the market. In this chapter we will discuss about various aspects of SQA.
  - Software Quality Assurance is the set of activities which ensure that the standards, processes and procedures are suitable for the project and implemented correctly.
  - Quality : Quality of Software is checked to see if it meets the requirements.
  - Assurance: It means ensuring the correctness of the results and security of the product, as it works without any bug and according to the expectations.
- 

# SOFTWARE RELIABILITY

- Software Reliability means Operational reliability. It is described as the ability of a system or component to perform its required functions under static conditions for a specific period.
- Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.
- Software Reliability is an essential connect of software quality, composed with functionality, usability, performance, serviceability, capability, install ability, maintainability, and documentation



# ISO 9000 QUALITY STANDARDS

- The ISO 9000 series was created by the International Organization for Standardization (ISO) as international requirements and guidelines for quality management systems.
- It was originally introduced in 1987 and over the years has established itself in the global economy having been adopted in over 178 countries with over one million registrations.
- The Current Version is ISO 9001: 2015 of the ISO 9001 Standard.



# WHY ISO

- ISO Standards are essential part of an societal institution.
- They Ensure quality and safety of our products and services in International Trade.
- Business can be seen to benefit from ISO standards as they can help cut costs by improved systems and procedures put in place.
- ISO 9001 is among ISO's best-known standards, and it defines the criteria for meeting a number of quality management principles. It helps businesses and organizations be more efficient and improve customer satisfaction.

