



CSE- 321

Software Engineering

Software Design

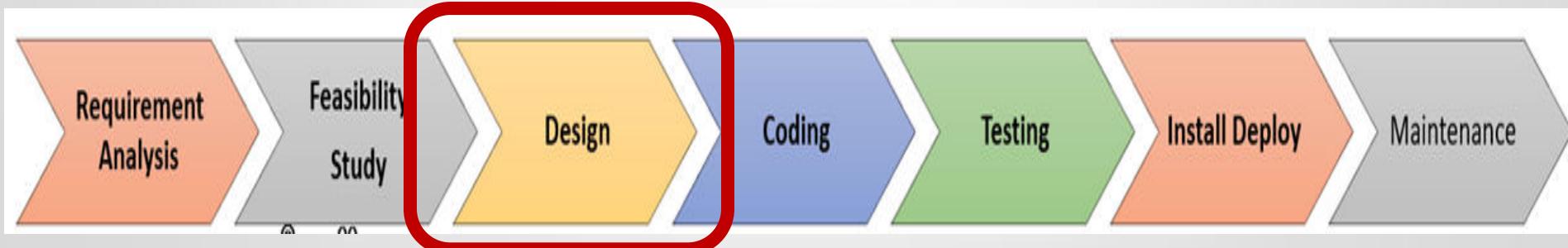
Lecture Outlines

- ✧ **Software design**
- ✧ **Design Principles**
- ✧ **Strategy of Design**
- ✧ **Coupling and Cohesion**
- ✧ **Architectural Design**
- ✧ **Abstract machine (layered) Design**
- ✧ **Distributed Systems Architectures**
- ✧ **Client-Server Architecture**
- ✧ **Broker Architectural Style : CORBA**
- ✧ **Service-Oriented Architecture (SOA)**

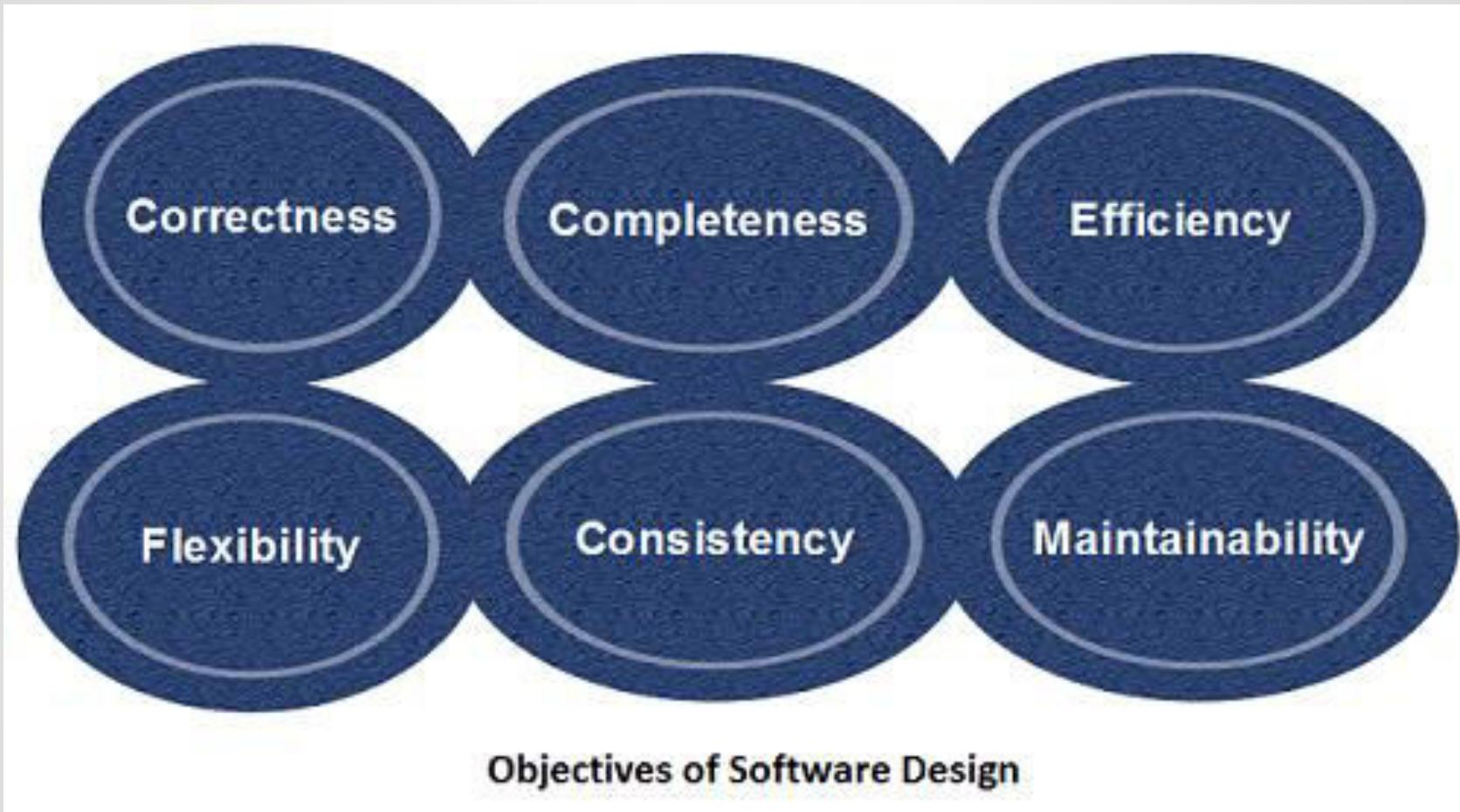
Software design

What is Software design ?

- ❖ Software design is a process to **transform user requirements** into some suitable form, which helps the programmer in **software coding and implementation**.
- ❖ Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from **problem domain to solution domain**. It tries to specify how to fulfill the requirements mentioned in SRS.



Objectives of Software Design



Objectives of Software Design

Objectives of Software Design

Following are the purposes of Software design:

- 1. Correctness:** Software design should be correct as per requirement.
- 2. Completeness:** The design should have all components like data structures, modules, and external interfaces, etc.
- 3. Efficiency:** Resources should be used efficiently by the program.
- 4. Flexibility:** Able to modify on changing needs.
- 5. Consistency:** There should not be any inconsistency in the design.
- 6. Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

Software Design Levels

Software design yields three levels of results:

Architectural Design

- Highest abstract version of the system.
- It identifies the software as a system with many components interacting with each other.
- Designers get the idea of proposed solution domain.

High-level Design

- Breaks the ‘single entity-multiple component’ concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other.

Detailed Design

- Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs

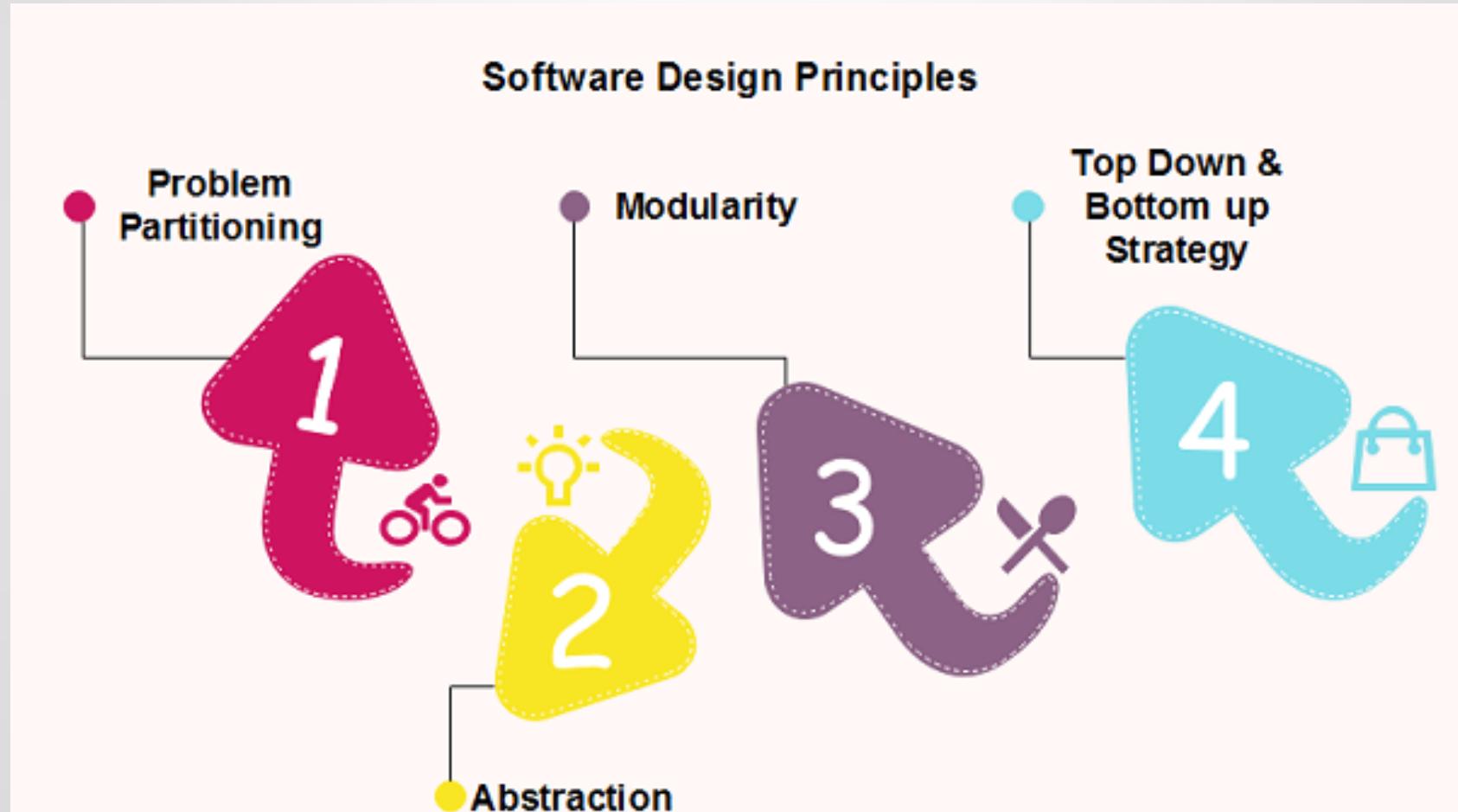
Software Design Principles

- Design is not coding, coding is not design.
- The design should be traceable to the analysis model.
- The design should not reinvent the wheel.
- The design should “minimize the intellectual distance” between the software and the problem as it exists in the real world.
- The design should exhibit uniformity and integration.
- The design should be structured to accommodate change.
- The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.

- The design should be assessed for quality as it is being created, not after the fact.
- The design should be reviewed to minimize conceptual (semantic) errors.

Software Design Principles

Software design principles are concerned with providing means to handle the complexity of the design process effectively.



Software Design Principles

Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

Benefits of Problem Partitioning

1. Software is easy to understand
2. Software becomes simple
3. Software is easy to test
4. Software is easy to modify
5. Software is easy to maintain
6. Software is easy to expand

As the number of partition increases = Cost of partition and complexity increases

Modularization

Modularization is a technique to divide a software system into multiple discrete and **independent modules**, which are expected to be capable of carrying out task(s) **independently**. These modules may work as basic constructs for the entire software.

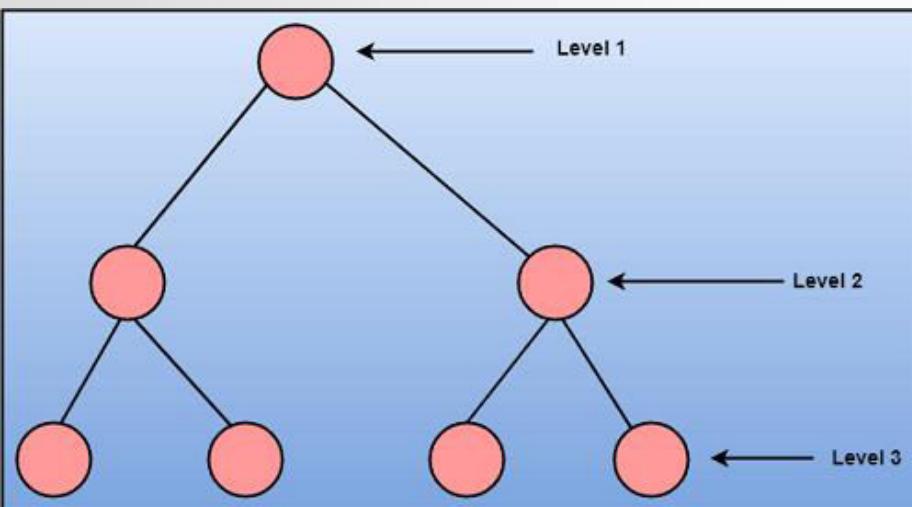
Advantage of modularization:

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- Concurrent execution can be made possible
- Desired from security aspect

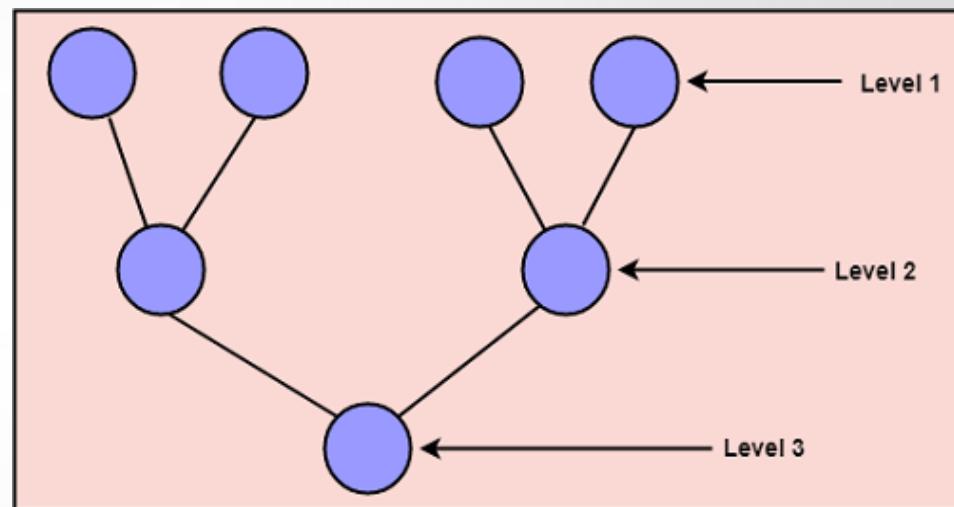
Strategy of Design

To design a system, there are two possible approaches:

1. Top-down Approach
2. Bottom-up Approach



Top-down Approach



Bottom-up Approach

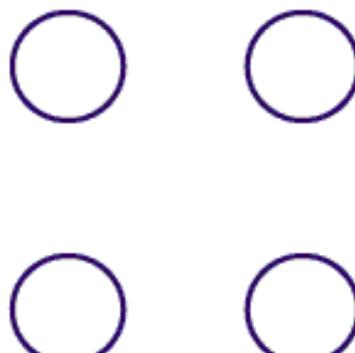
Coupling and Cohesion

Module Coupling

In software engineering, Coupling is measured by the **number of relations between the modules**. That means, the coupling is the degree of interdependence between software modules.

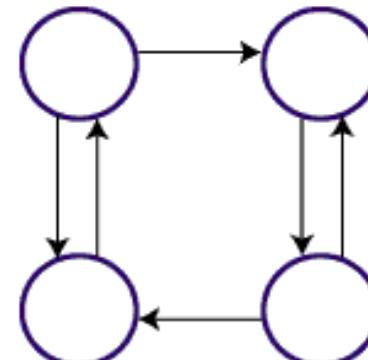
A good design is the one that has low coupling.

Module Coupling



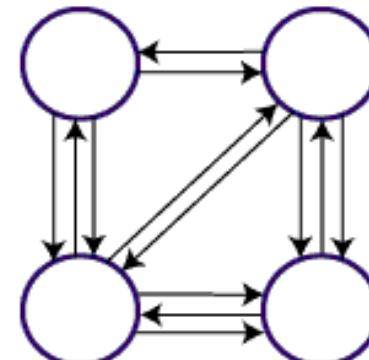
Uncoupled: no
dependencies

(a)



Loosely Coupled:
Some dependencies

(b)



Highly Coupled:
Many dependencies

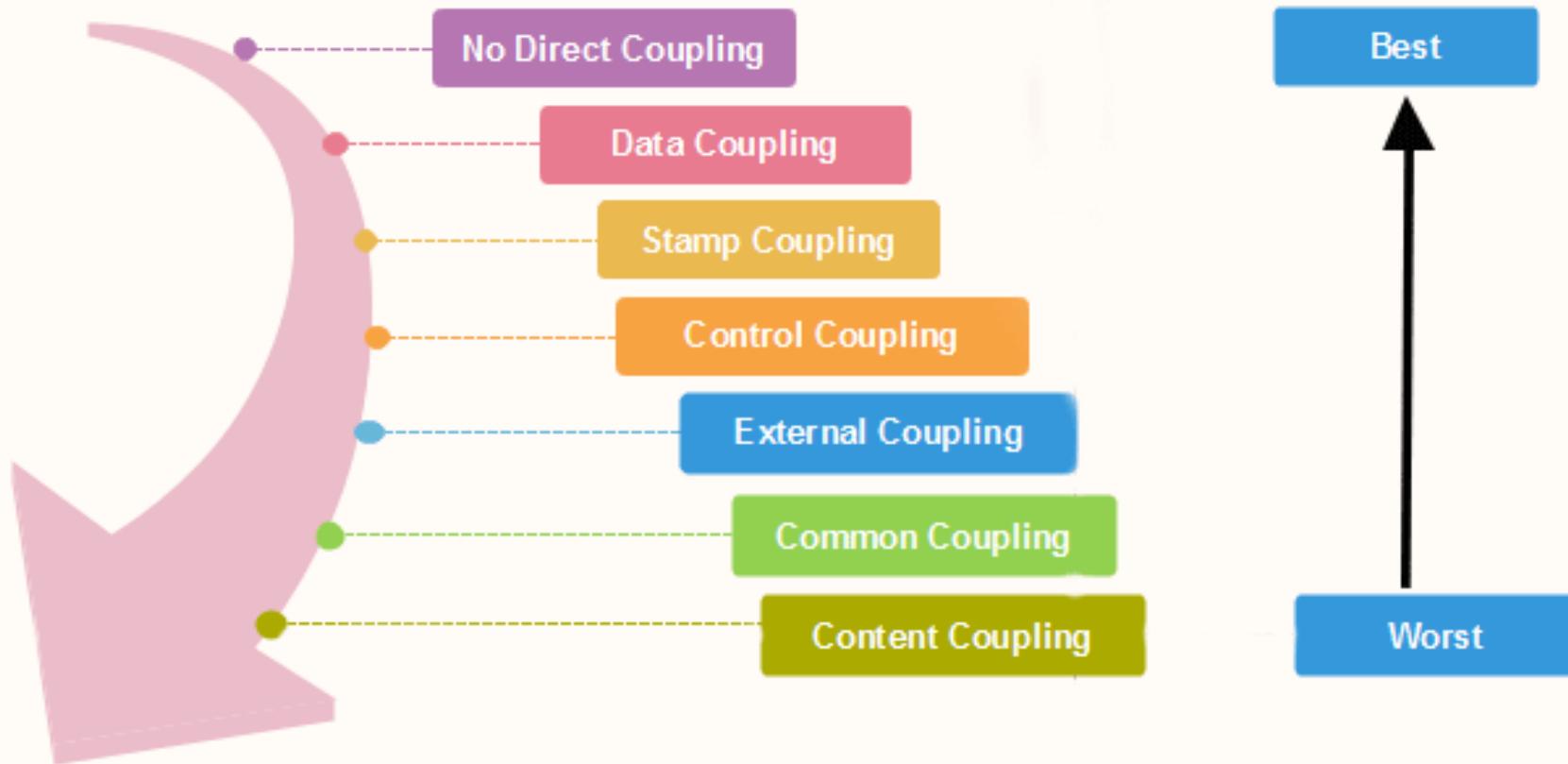
(c)

Coupling and Cohesion

Types of Module Coupling

Types of Modules Coupling

There are various types of module Coupling are as follows:



Coupling and Cohesion

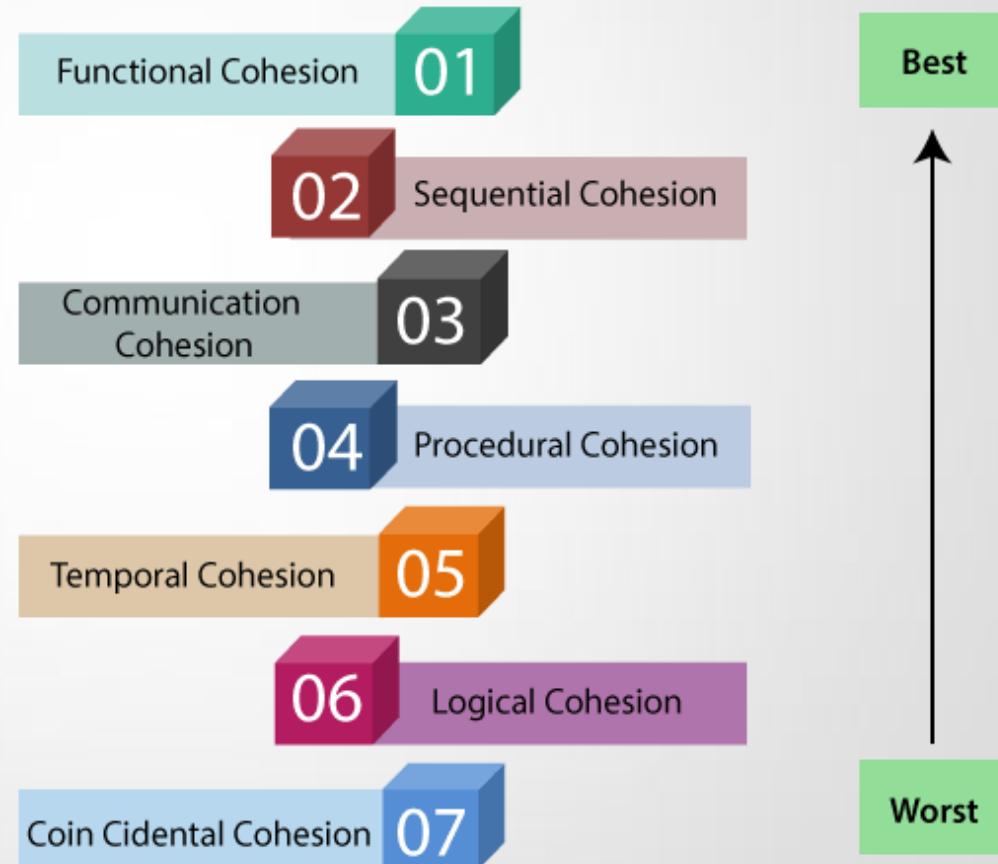
Module Cohesion

cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the **strength of relationships between pieces of functionality within a given module.**

Basically, cohesion is the internal glue that keeps the module together.

A good software design will have high cohesion.

Types of Modules Cohesion



Coupling vs Cohesion

Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative independence between the modules.	Cohesion shows the module's relative functional strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

Design Pattern

- ❖ A design pattern provides a general reusable solution for the common problems occurs in software design.
- ❖ The patterns typically **show relationships and interactions between classes or objects.**
- ❖ The idea is to speed up the development process by providing well tested, proven development/design paradigm. Design patterns are programming language independent strategies for solving a common problem.
- ❖ That means a design pattern represents an idea, not a particular implementation. By using the design patterns you can make your code more flexible, reusable and maintainable.

Design Pattern

❖ Types of Design Patterns

There are mainly three types of design patterns:

1. Creational Design Pattern

- Factory Pattern
- Abstract Factory Pattern
- Singleton Pattern
- Prototype Pattern
- Builder Pattern.

2. Structural Design Pattern

- Adapter Pattern
- Bridge Pattern
- Composite Pattern
- Decorator Pattern
- Facade Pattern
- Flyweight Pattern
- Proxy Pattern

3. Behavioural Design Pattern

- Chain Of Responsibility Pattern
- Command Pattern
- Interpreter Pattern
- Iterator Pattern
- Mediator Pattern
- Memento Pattern
- Observer Pattern
- State Pattern
- Strategy Pattern
- Template Pattern
- Visitor Pattern

Design Pattern

Difference Between Architectural Style, Architectural Patterns

Architectural Style

The architectural style shows how do we organize our code, or how the system will look like from 10000 feet helicopter view to show the highest level of abstraction of our system design. Furthermore, when building the architectural style of our system we focus on **layers** and **modules** and how they are communicating with each other.

Architectural Patterns

The architectural pattern shows how a solution can be used to solve a reoccurring problem. In another word, it reflects how a code or components **interact** with each other

Architectural Design

Architectural Design

- Architectural design is a process for identifying the sub-systems making up a system and the framework for sub-system control and communication.
- The output of this design process is a description of the **software architecture**.
- It represents the link between specification and design processes
- Often carried out in **parallel** with some specification activities.
- It involves identifying major system **components** and their **communications**.

Architectural Design

- A software architecture is a description of how a **software system is organized**.
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used.
- Architectures may be **documented from several different perspectives** or views such as a conceptual view, a logical view, a process view, and a development view.
- Architectural patterns are a means of reusing knowledge about generic system architectures.
- Architectural patterns are similar to software design pattern but have a broader scope.

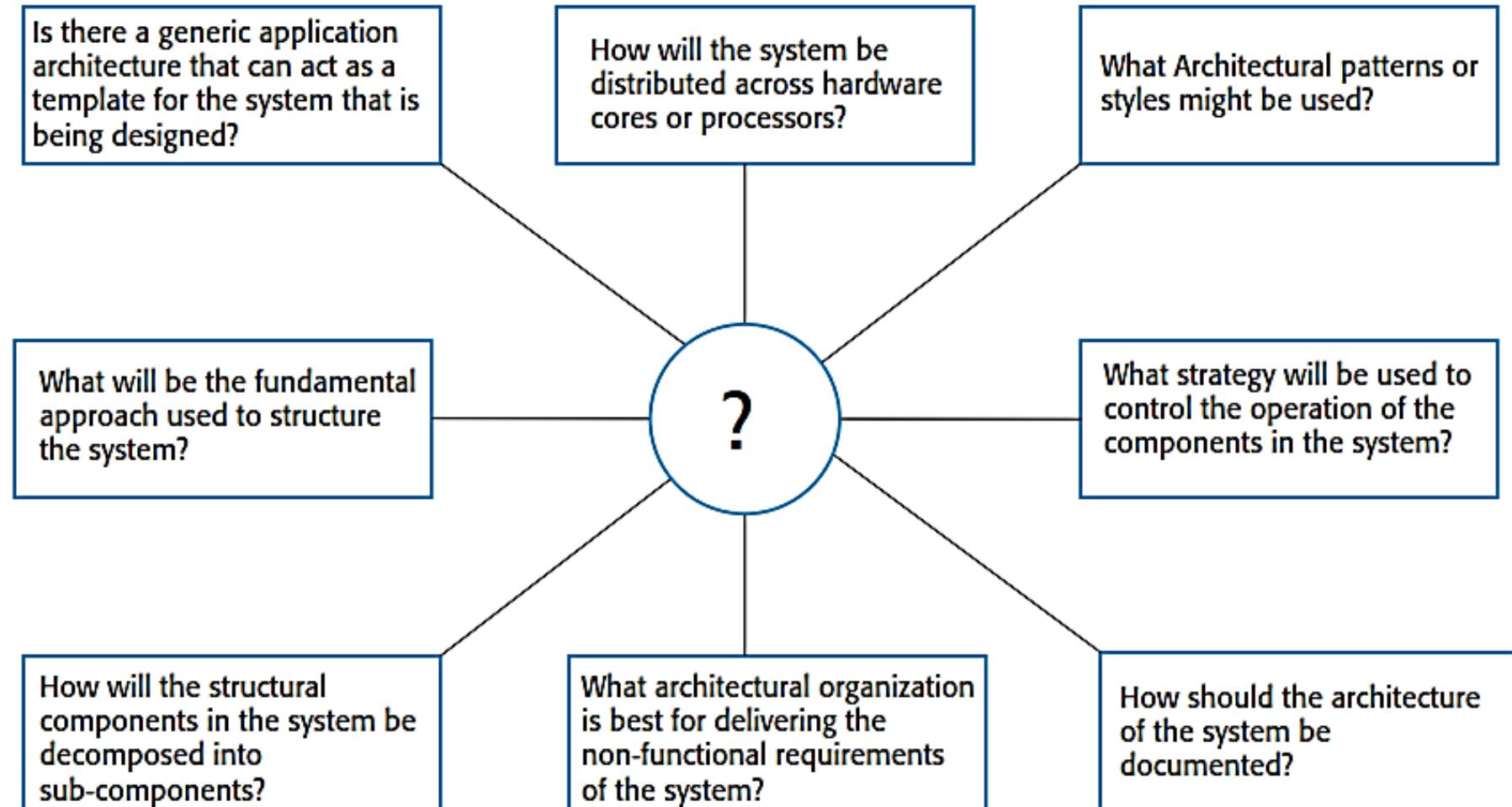
Architectural design decisions

Architectural design is a **creative process** so the process differs depending on the type of system being developed.

However, a number of **common decisions** span all design processes.

- Is there a **generic** application architecture that can be used?
- How will the system be **distributed**?
- What architectural **styles** are appropriate?
- What approach will be used to **structure** the system?
- How will the system be **decomposed** into modules?
- What **control strategy** should be used?
- How will the architectural design be **evaluated**?
- How should the architecture be **documented**?

Architectural design decisions



4 + 1 view model of software architecture

- A **logical view**, which shows the key abstractions in the system as objects or object classes.
- A **process view**, which shows how, at run-time, the system is composed of interacting processes.
- A **development view**, which shows how the software is decomposed for development.
- A **physical view**, which shows the system hardware and how software components are distributed across the processors in the system.
- ***Related using use cases or scenarios (+1)***

Architectural patterns

- Patterns are a means of representing, sharing and reusing knowledge.
- An architectural pattern is a **stylized description** of good design practice, which has been tried and tested in different environments.
- Patterns should include **information about** when **they** are and when **they are not useful**.
- Patterns may be represented using tabular and graphical descriptions.

The Model-View-Controller (MVC) pattern

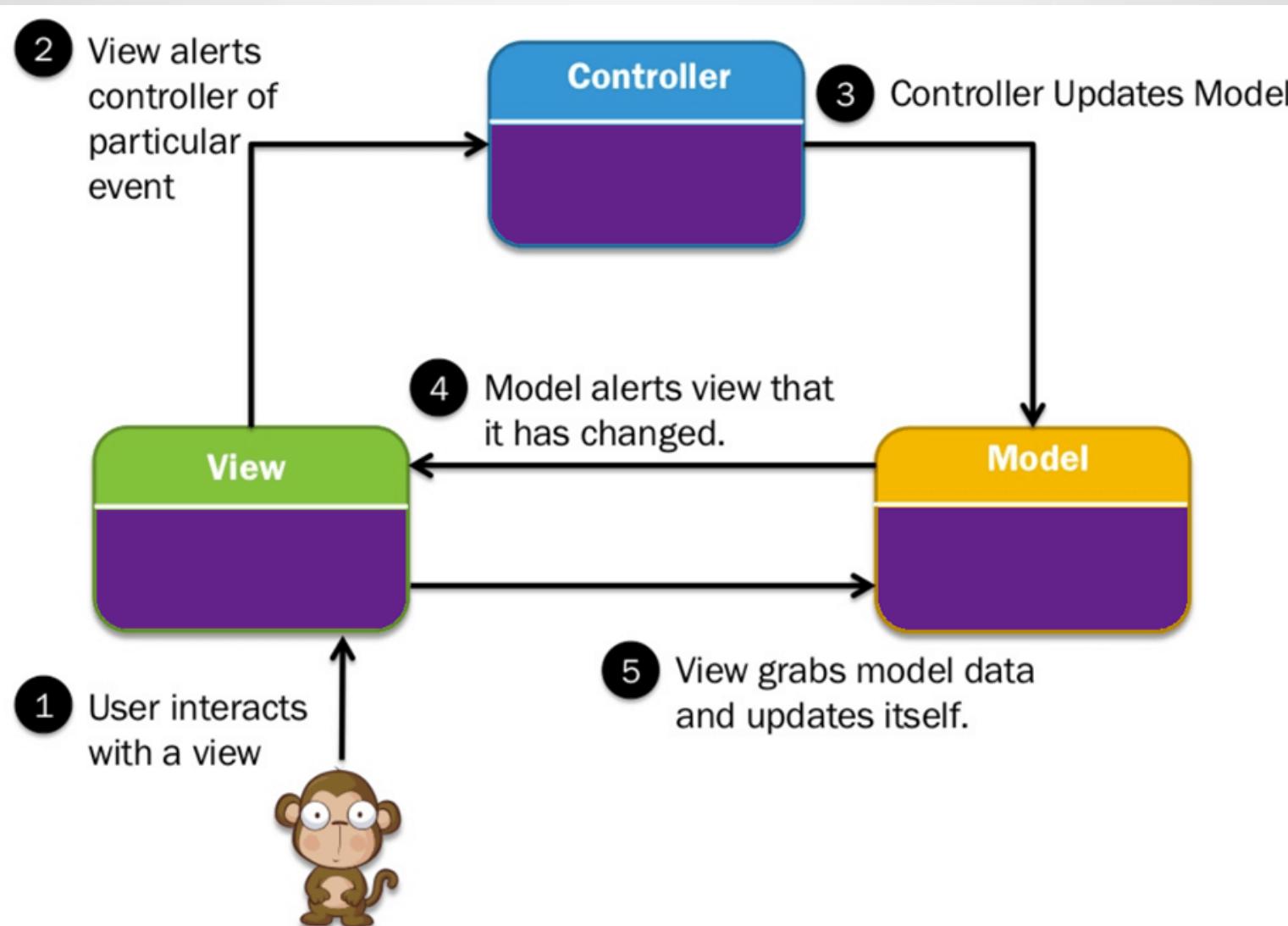
- **MVC** (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic.
- It emphasizes a separation between the software's business logic and display.
- This "separation of concerns" provides for a better division of labour and improved maintenance.
- Some other design patterns are based on MVC, such as
 - **MVVM (Model-View-Viewmodel),**
 - **MVP (Model-View-Presenter), and**
 - **MVW (Model-View-Whatever).**

The Model-View-Controller (MVC) pattern

- Serves as a basis of interaction management in many web-based systems.
- Decouples three major interconnected components:
 - The **model** is the central component of the pattern that directly manages the data, logic and rules of the application. It is the application's dynamic data structure, independent of the user interface.
 - A **view** can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible.
 - The **controller** accepts input and converts it to commands for the model or view.
- Supported by most language frameworks.

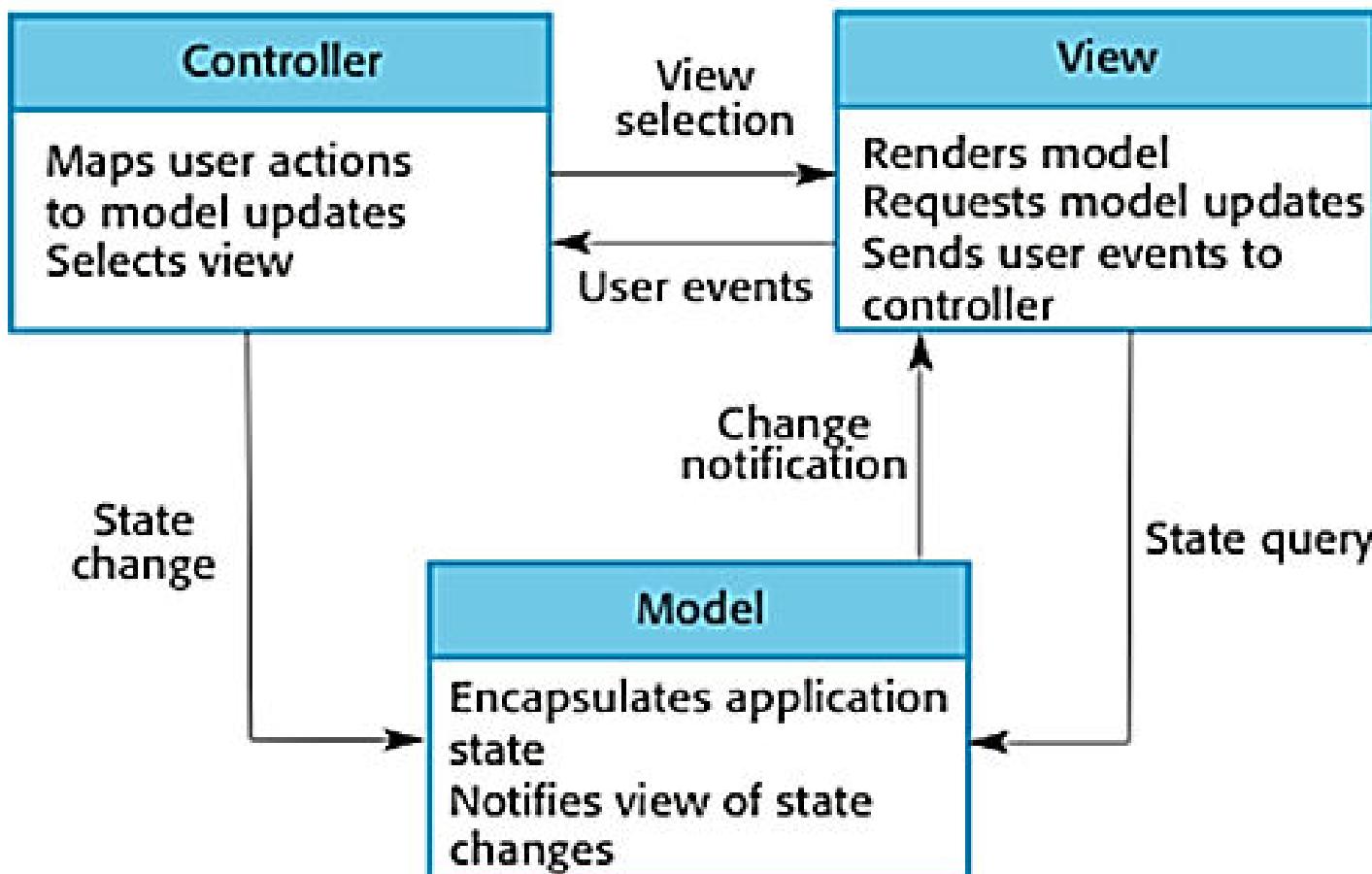
The Model-View-Controller (MVC) pattern

MVC architectural pattern



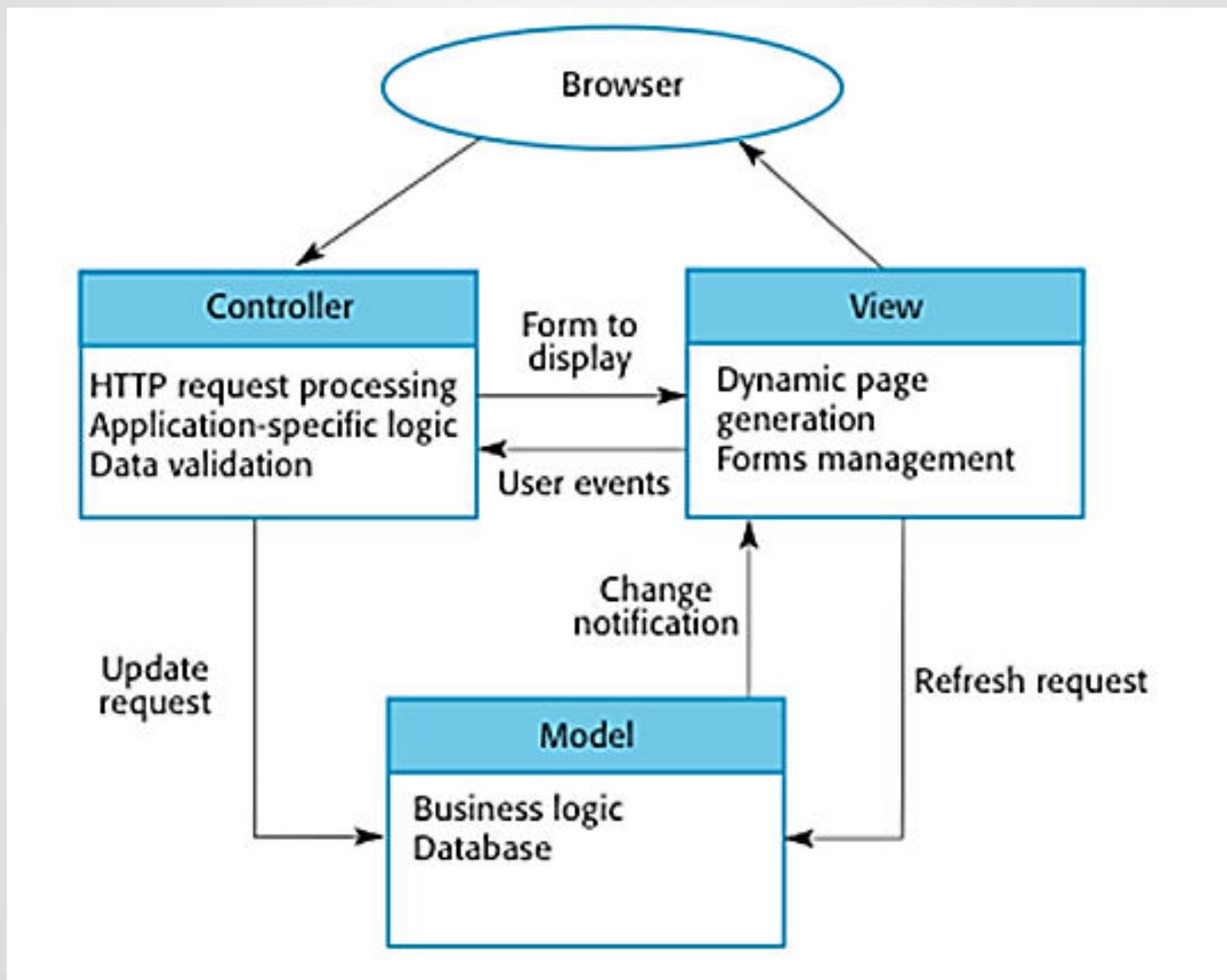
The Model-View-Controller (MVC) pattern

MVC architectural pattern



The Model-View-Controller (MVC) pattern

Web application architecture using the MVC pattern



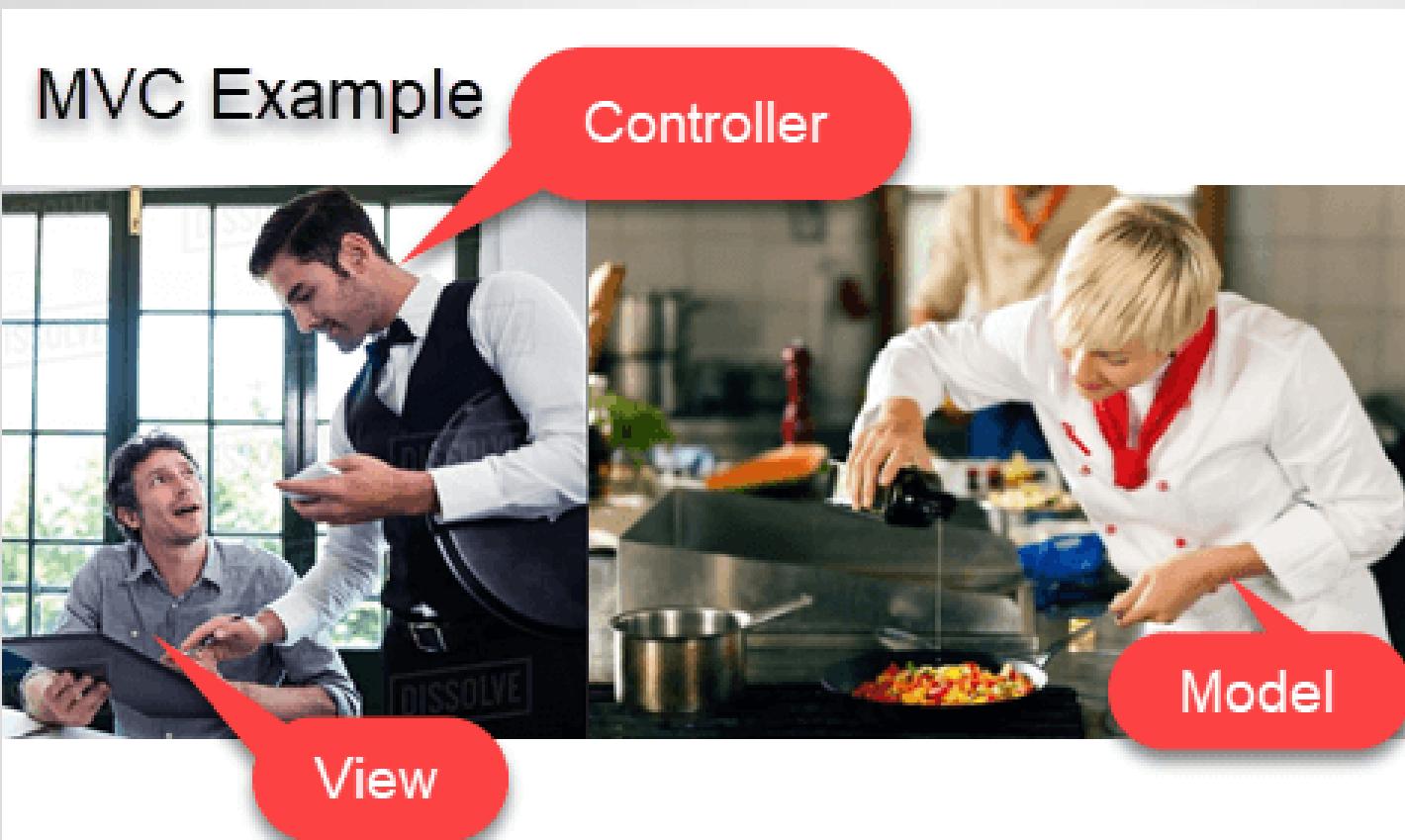
The Model-View-Controller (MVC) pattern

Case- Study : restaurant

1. Let's assume you go to a restaurant. You will not go to the kitchen and prepare food which you can surely do at your home. Instead, you just go there and wait for the waiter to come on.
2. Now the waiter comes to you, and you just order the food. The waiter doesn't know who you are and what you want he just written down the detail of your food order.
3. Then, the waiter moves to the kitchen. In the kitchen waiter not prepare your food.
4. The cook prepares your food. The waiter is given your order to him along with your table number.
5. Cook then prepared food for you. He uses ingredients to cooks the food. Let's assume that your order a vegetable sandwich. Then he needs bread, tomato, potato, capsicum, onion, bit, cheese, etc. which he sources from the refrigerator
6. Cook final hand over the food to the waiter. Now it is the job of the waiter to moves this food outside the kitchen.
7. Now waiter knows which food you have ordered and how they are served.

The Model-View-Controller (MVC) pattern

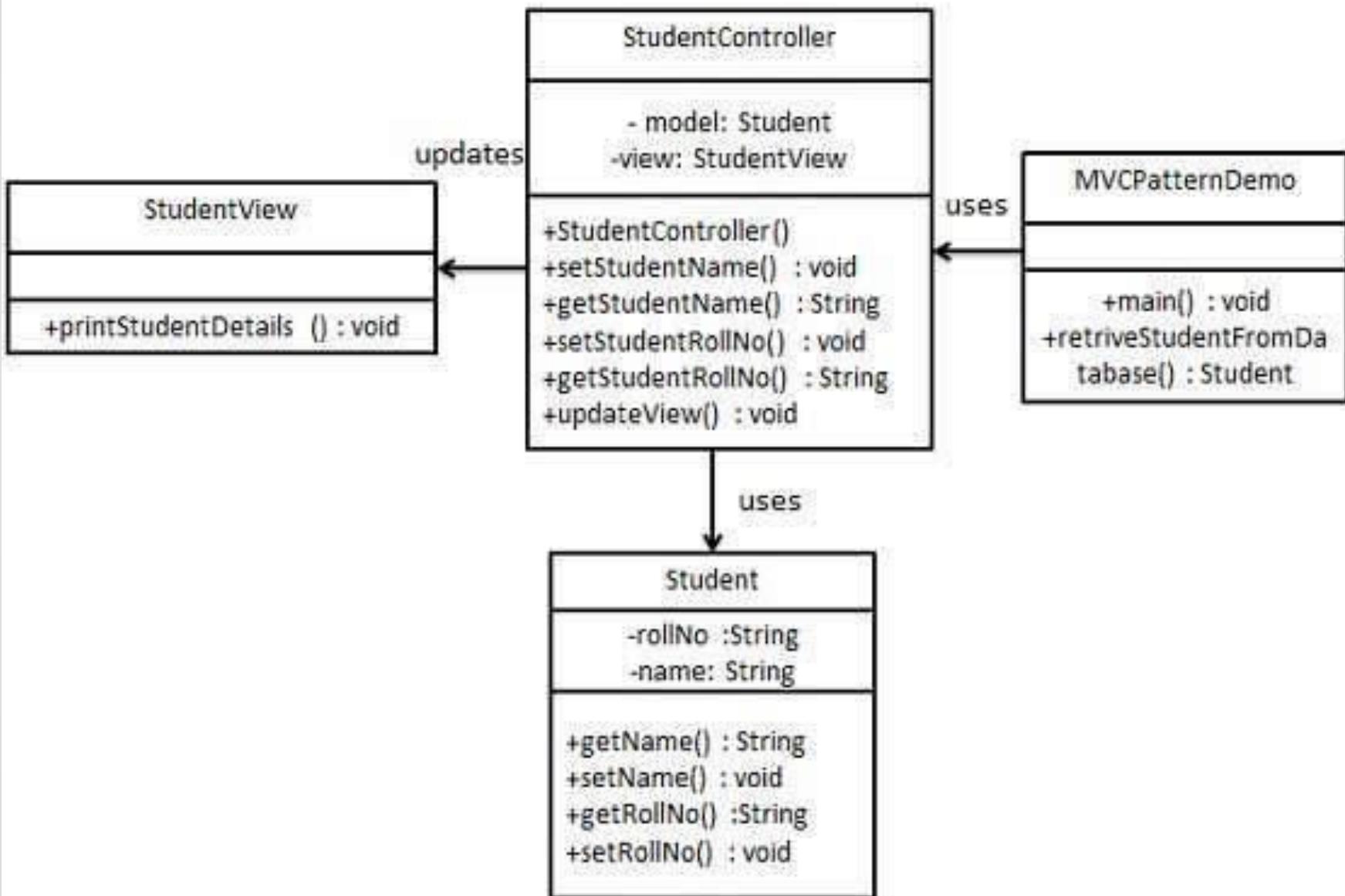
Case- Study : restaurant



In this case,

You= View
Waiter= Controller
Cook= Model
Refrigerator= Data

Coding Style : MVC



Design Patterns - MVC

Step 1
Create Model

```
public class Student {  
    private String rollNo;  
    private String name;  
  
    public String getRollNo() {  
        return rollNo;  
    }  
  
    public void setRollNo(String rollNo) {  
        this.rollNo = rollNo;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

Design Patterns - MVC

Step 2

Create view

```
public class StudentView {  
    public void printStudentDetails(String studentName, String studentRollNo){  
        System.out.println("Student: ");  
        System.out.println("Name: " + studentName);  
        System.out.println("Roll No: " + studentRollNo);  
    }  
}
```

Design Patterns - MVC

Step 3
Create Controller

```
public class StudentController {  
    private Student model;  
    private StudentView view;  
  
    public StudentController(Student model, StudentView view){  
        this.model = model;  
        this.view = view;  
    }  
  
    public void setStudentName(String name){  
        model.setName(name);  
    }  
    public String getStudentName(){  
        return model.getName();  
    }  
    public void setStudentRollNo(String rollNo){  
        model.setRollNo(rollNo);  
    }  
    public String getStudentRollNo(){  
        return model.getRollNo();  
    }  
  
    public void updateView(){  
        view.printStudentDetails(model.getName(), model.getRollNo());  
    }  
}
```

Design Patterns - MVC

Step 4
MVCPatternDemo

```
public class MVCPatternDemo {  
    public static void main(String[] args) {  
  
        //fetch student record based on his roll no from the database  
        Student model = retriveStudentFromDatabase();  
  
        //Create a view : to write student details on console  
        StudentView view = new StudentView();  
  
        StudentController controller = new StudentController(model, view);  
  
        controller.updateView();  
  
        //update model data  
        controller.setStudentName("John");  
  
        controller.updateView();  
    }  
  
    private static Student retriveStudentFromDatabase(){  
        Student student = new Student();  
        student.setName("Robert");  
        student.setRollNo("10");  
        return student;  
    }  
}
```

In Python
https://www.tutorialspoint.com/python_design_patterns/python_design_patterns_model_view_controller.htm

Design Patterns - MVC

Step 5

Verify the output.

Student:

Name: Robert

Roll No: 10

What is MVVM?

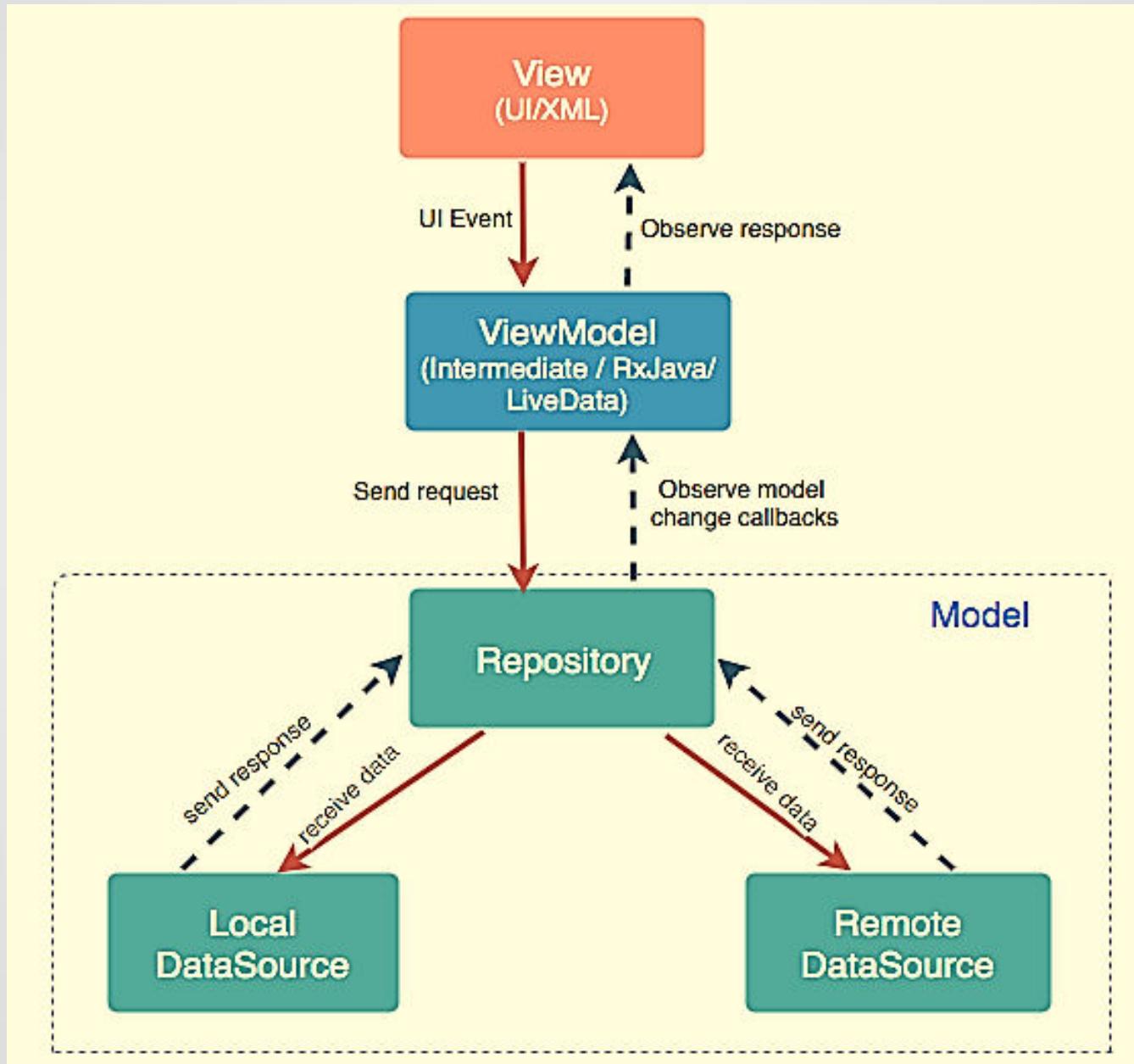
MVVM architecture facilitates a separation of development of the graphical user interface with the help of mark-up language or GUI code.

The full form of MVVM is Model–View–ViewModel.

MVVM removes the tight coupling between each component.

Most importantly, in this architecture, the children don't have the direct reference to the parent, they only have the reference by observables.

MVVM



Model: It represents the **data and the business logic** of the Application. It consists of the business logic - **local and remote data source, model classes, repository**.

View: It consists of the **UI Code** (Activity, Fragment), XML. It sends the user action to the ViewModel but does **not** get the response back directly. To get the response, it has to subscribe to the observables which ViewModel exposes to it.

ViewModel: It is a **bridge** between the View and Model(business logic). It does not have any clue which View has to use it as it does not have a direct reference to the View. So basically, the ViewModel should not be aware of the view who is interacting with. It interacts with the Model and exposes the observable that can be observed by the View.

Layered architecture

Have you ever wondered how Google makes Gmail work in different languages all over the world? Users can use Gmail every day in English, Spanish, French, Russian, and many more languages.

Did Google develop different Gmail applications for each country?
Of course not.

They developed an internal version that does all the message processing, and then developed different external user interfaces that work in many languages.

Layered architecture

Google developed the Gmail application in **different layers**:

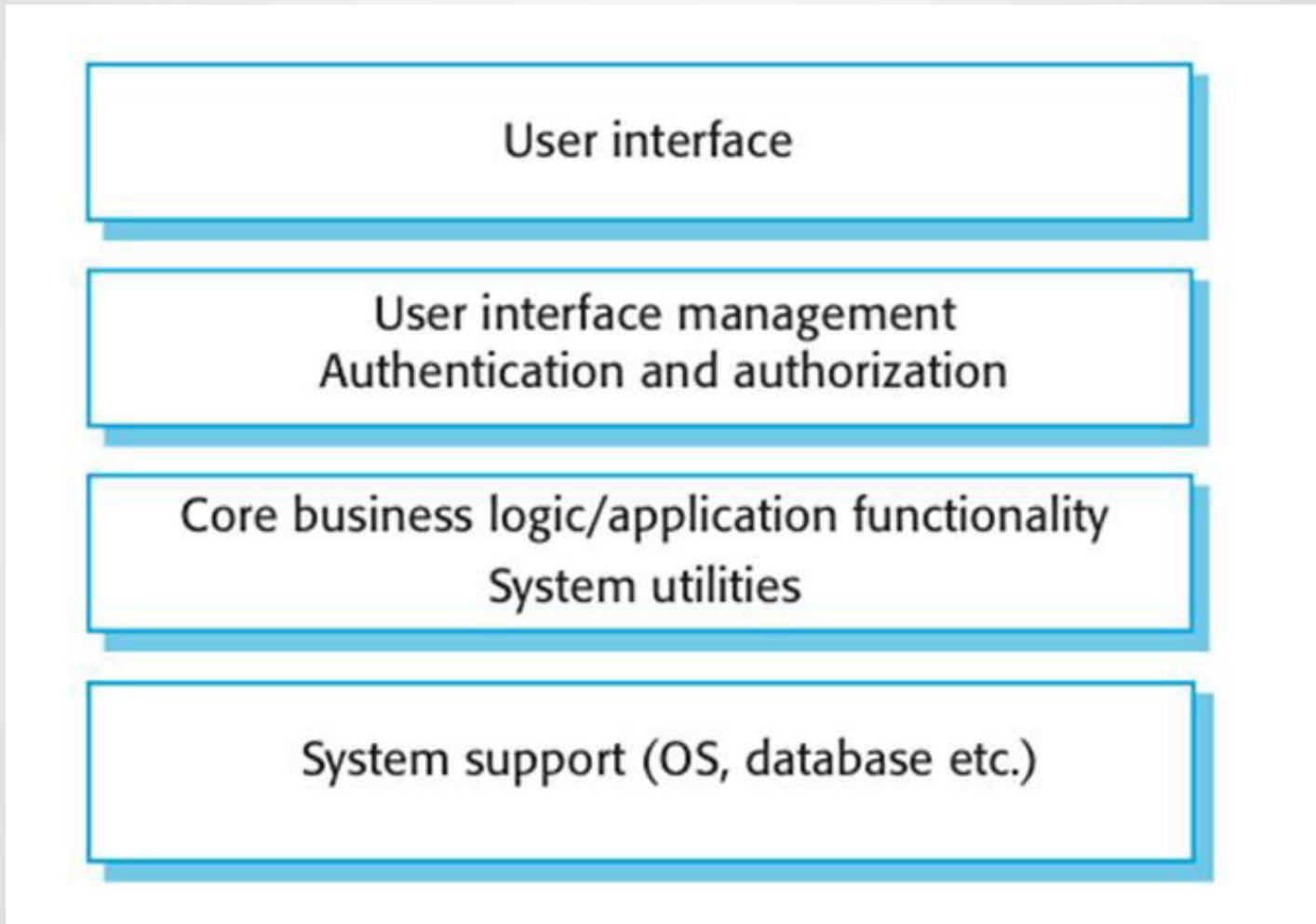
1. There is an **internal layer** that does all the processing.
2. There is an **external layer** that communicates with the users in their language.
3. There is also **another layer that interacts with a database** where user email messages are stored (millions or maybe billions).

Gmail is divided into at least three layers, every one of them has a mission, and they exist separately to handle different processes at different levels. It is an excellent example of a **layered architecture**.

Layered architecture

- Used to model the **interfacing of sub-systems**.
- Organizes the system into a set of layers (or abstract machines) each of which **provide a set of services**.
- Supports **the incremental development** of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artificial to structure systems in this way.

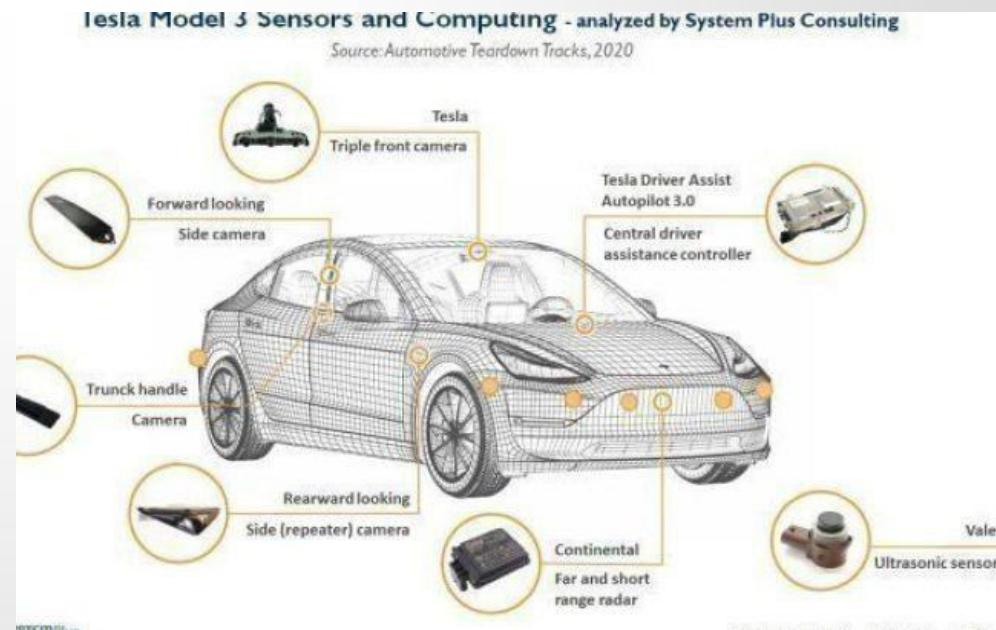
A generic layered architecture



Layered architecture

Real-World Example

- Tesla Control Module ([tesla.com](https://www.tesla.com)): Control system for self- driving vehicles
- Waymo software ([waymo.com](https://www.waymo.com))



Case-Study: Solving a Business Problem With Layered Architecture

Amaze is a project management software company.

Their product is sold globally with a monthly pay-per-user model and widely known among the project management community for being easy to use and able to operate on many different devices (PCs, Notebooks, laptops, tablets, iPhones, iPads, and Android phones).

Case-Study: Solving a Business Problem With Layered Architecture

What's the Business Problem?

The business problem is very straightforward: Amaze must work on any popular device on the market and be **able to support future devices**.

There must be **only one version** of the software for all devices. No special cases, no exceptions allowed.

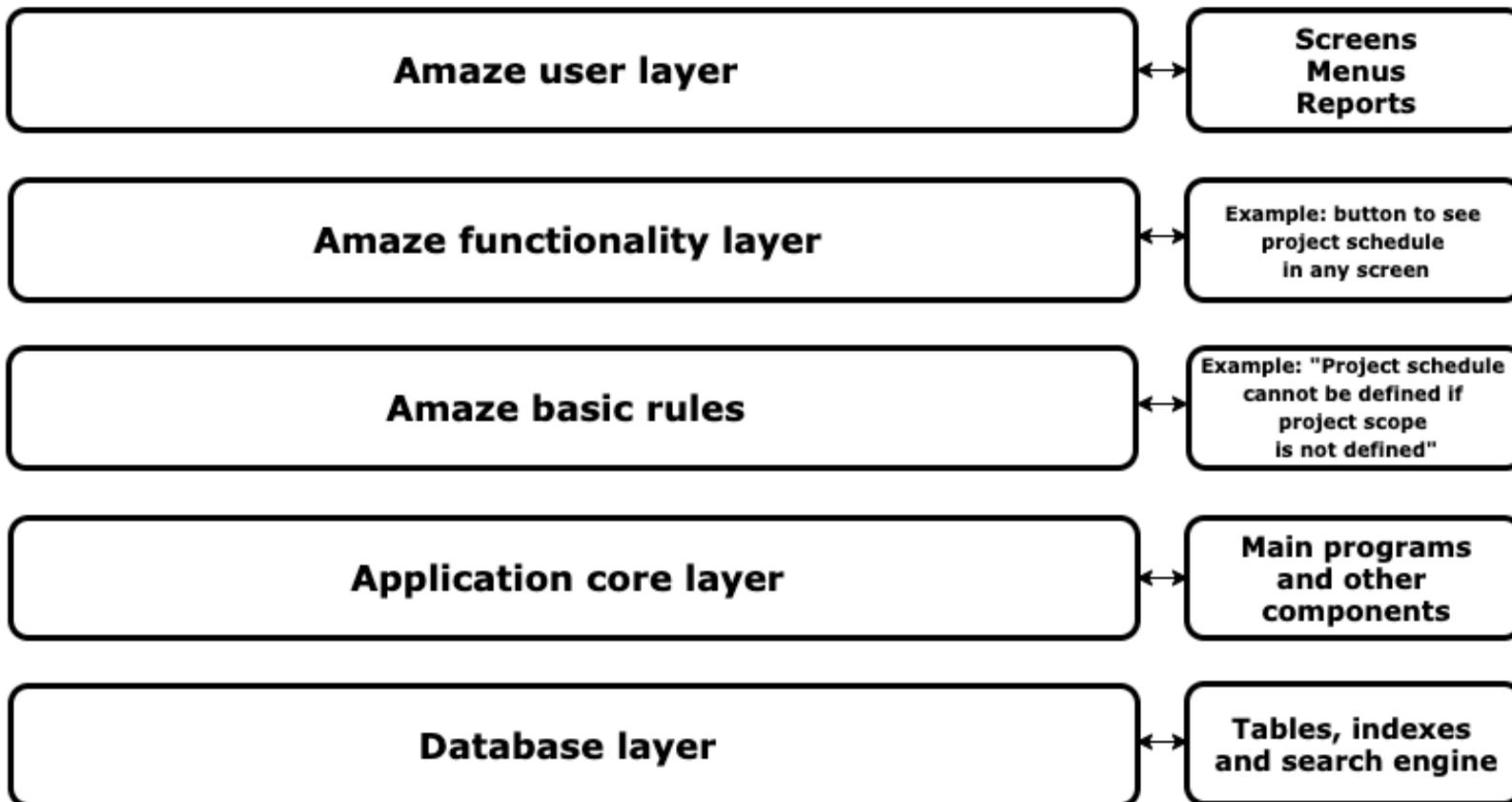
So to sum up:

- We know that **users have different devices**.
- There must be **only one software application** because the company wants to have low software maintenance costs.
- When **new device launches**, we do not want to change the whole software product.

Case-Study: Solving a Business Problem With Layered Architecture

What's the Solution?

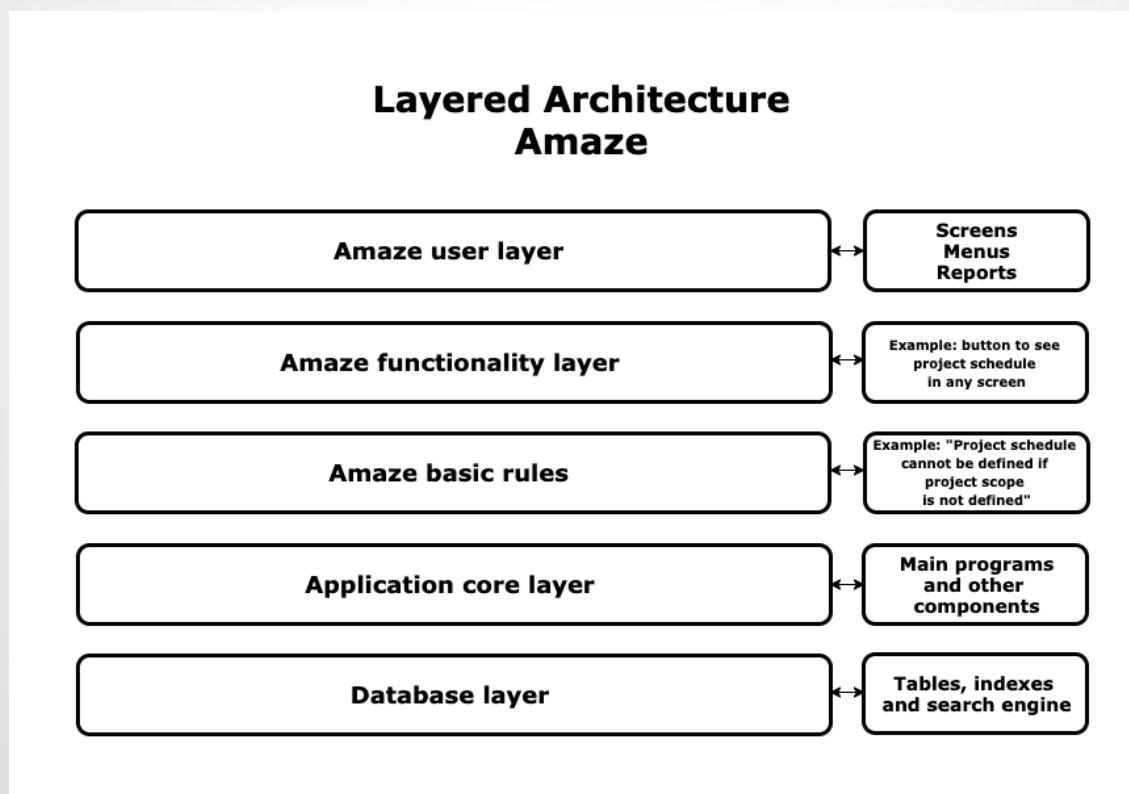
Layered Architecture Amaze



Case-Study: Solving a Business Problem With Layered Architecture

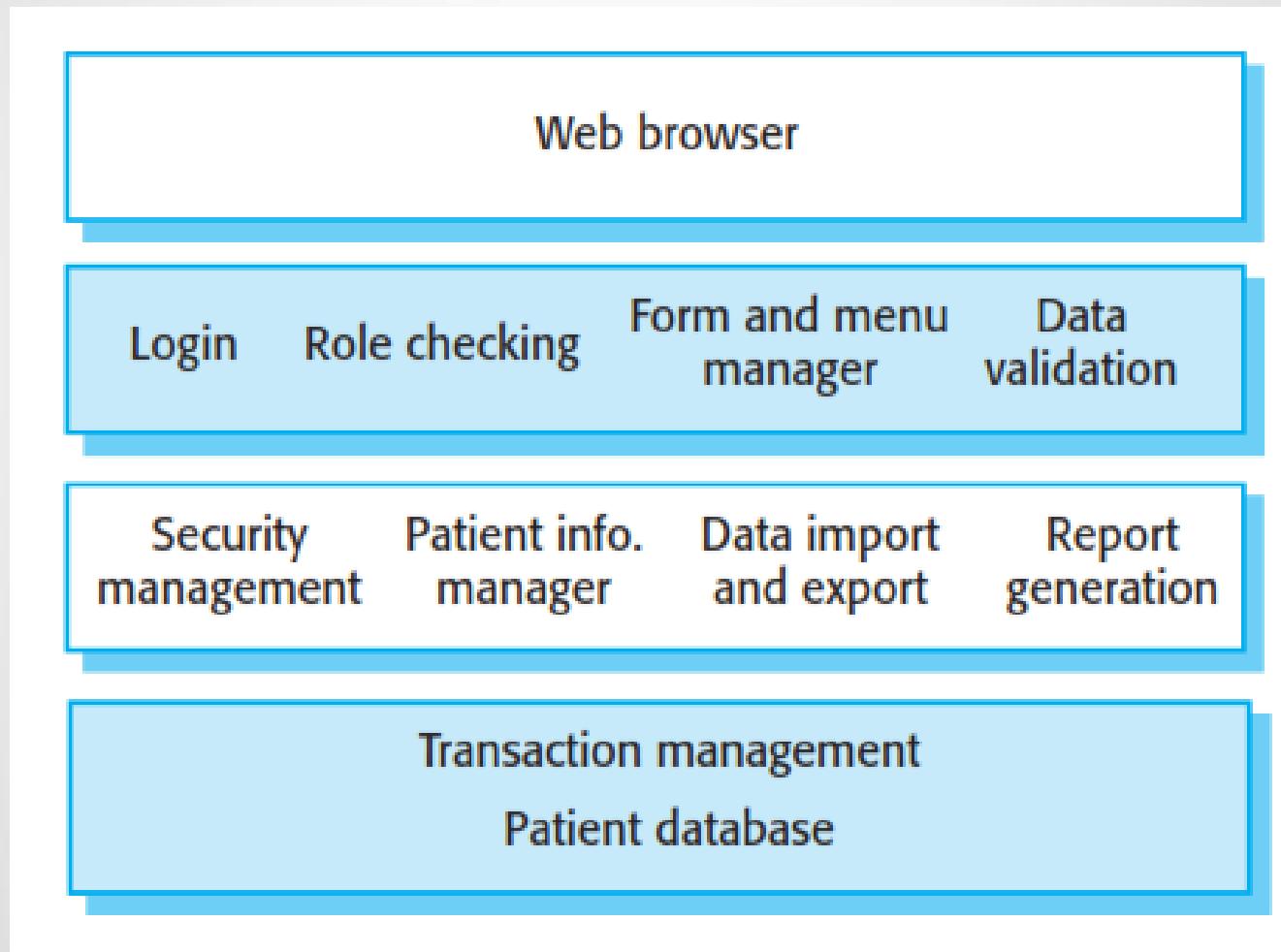
In our case, the **Amaze basic rules layer** is **critical**. This layer contains rules that determine the **behaviour of the whole application**, such as, "You can create a project schedule only if the project scope is defined."

The product's intelligence is in this layer: all special features that come from decades of experience in projects are developed there. The **application core layer** will be the most significant part of the application code.



The architecture of the Mentcare system

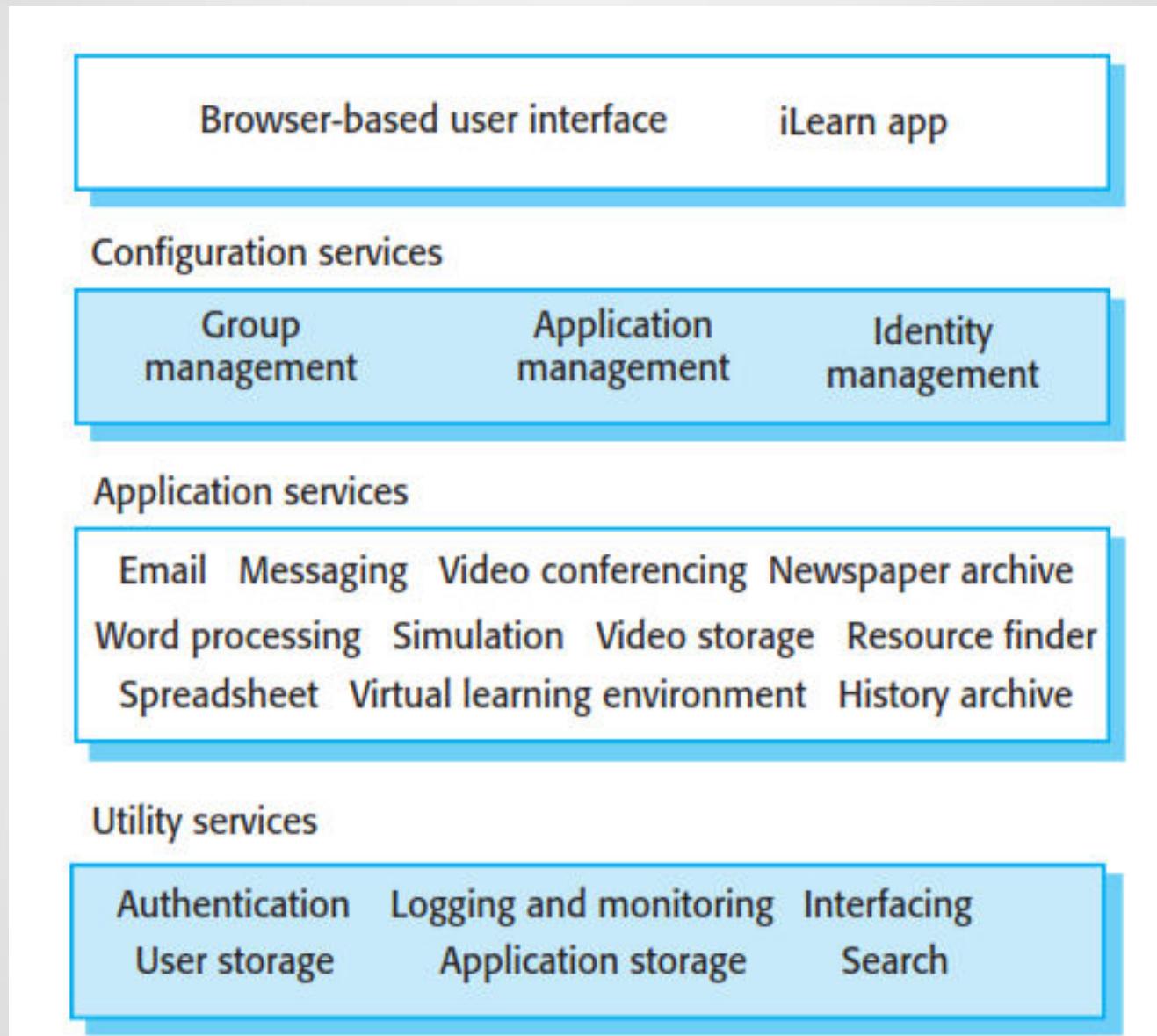
This system maintains and manages details of patients who are consulting specialist doctors about mental health problems.



The architecture of the Mentcare system

1. The top layer is a **browser-based user interface**.
2. The second layer provides the **user interface functionality** that is delivered through the web browser. It includes components to allow users to **log-in** to the system and checking components that ensure that the operations they use are allowed by their role. This layer includes form and menu management components that present information to users, and data validation components that check information consistency.
3. The third layer implements the functionality of the system and provides components that implement **system security**, patient information creation and updating, import and export of patient data from other databases, and report generators that create management reports.
4. Finally, the lowest layer, which is built using a **commercial database management system**, provides transaction management and persistent data storage.

The architecture of the iLearn system



Layered architecture

There are several **advantages** to using layered architecture:

- **Layers are autonomous**: A group of changes in one layer does not affect the others. This is good because we can increase the functionality of a layer, for example, making an application that works only on PCs to work on phones and tablets, without having to rewrite the whole application.
- Layers allow **better system customization**.

There are also a few key **disadvantages**:

- Layers make an application more difficult to maintain. **Each change requires analysis**.
- Layers may affect application performance because they create overhead in execution: each layer in the upper levels must connect to those in the lower levels for each operation in the system.

Control styles

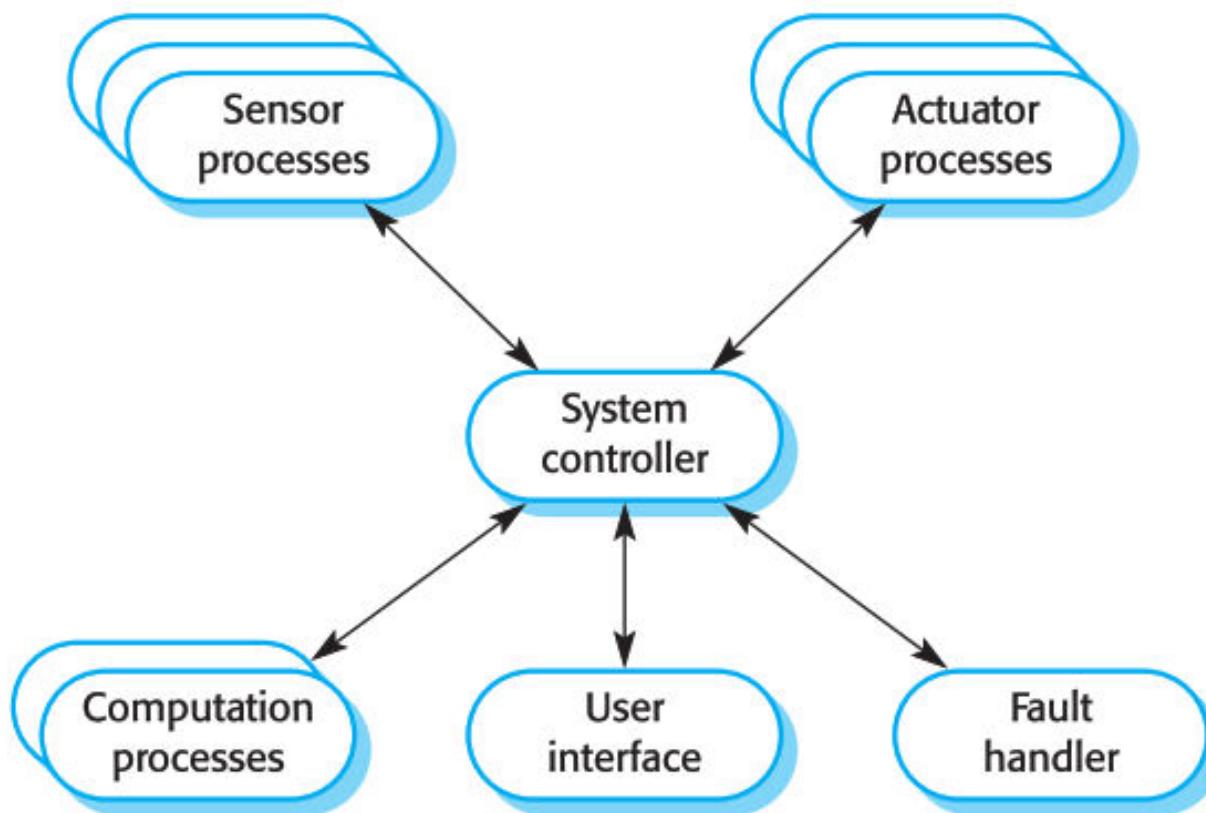
- Are concerned with the control flow between sub-systems.
Distinct from the system decomposition model.
- **Centralised control**
 - One sub-system has overall responsibility for control and starts and stops other sub-systems.
- **Event-based control**
 - Each sub-system can respond to externally generated events from other sub-systems or the system's environment.

Centralized control

- A control sub-system takes responsibility for managing the execution of other sub-systems.
- Call-return model
 - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems.
- Manager model
 - Applicable to concurrent systems. One system component controls the stopping, starting and coordination of other system processes. Can be implemented in sequential systems as a case statement.

Centralised control

A centralized control model for a real-time system



The repository model

Sub-systems must **exchange data**. This may be done in two ways:

- Shared data is held in a **central database** or repository and may be accessed by all sub-systems;
- Each sub-system maintains its **own database** and passes data explicitly to other sub-systems.

When large amounts of data are to be shared, the repository model of sharing is most commonly used.

The repository model

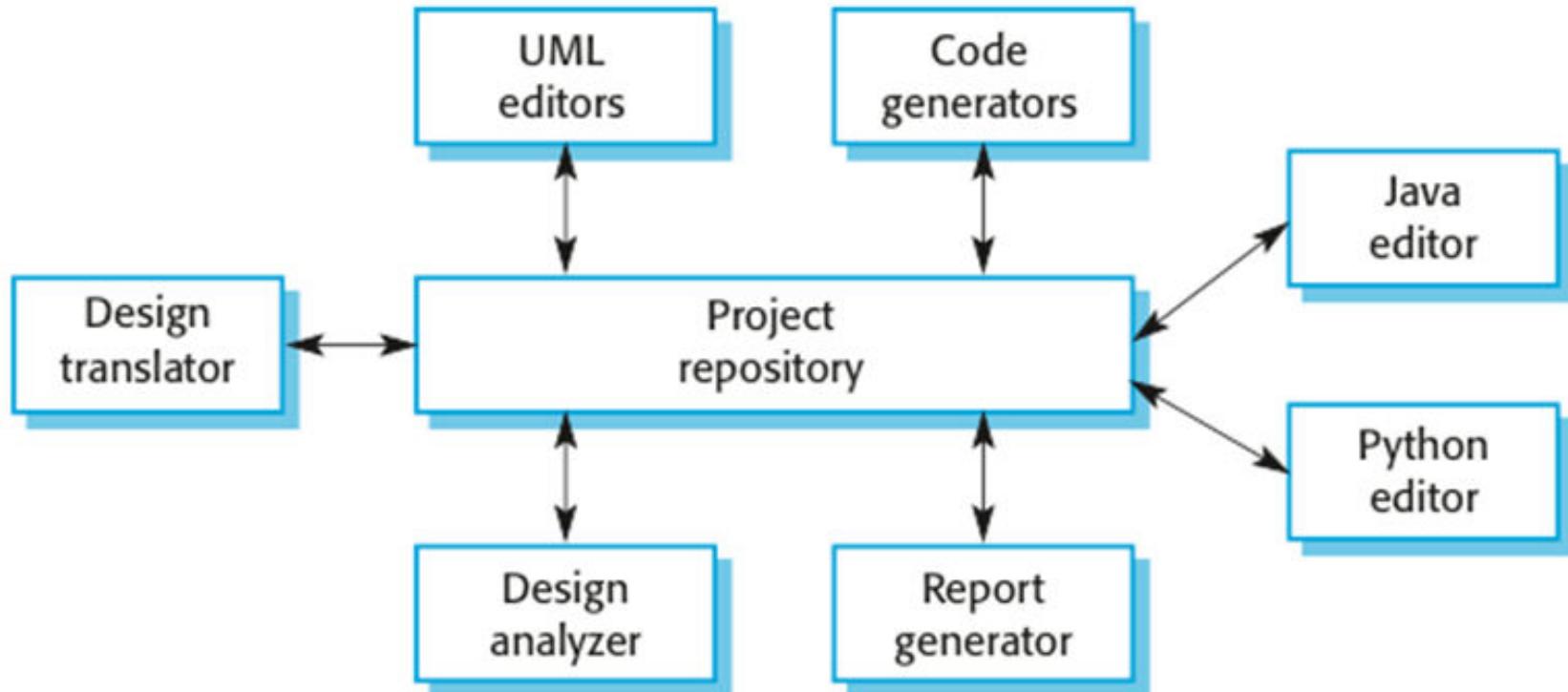
Advantages

- Efficient way to share **large amounts** of data;
- Sub-systems need not be concerned with how data is **produced**
- **Centralised** management e.g. backup, security, etc.
- **Sharing** model is published as the repository schema.

Disadvantages

- Sub-systems must agree on a repository data model. Inevitably a **compromise**;
- **Data evolution** is difficult and expensive;
- No scope for **specific** management policies;
- Difficult to **distribute** efficiently.

A repository architecture for an IDE



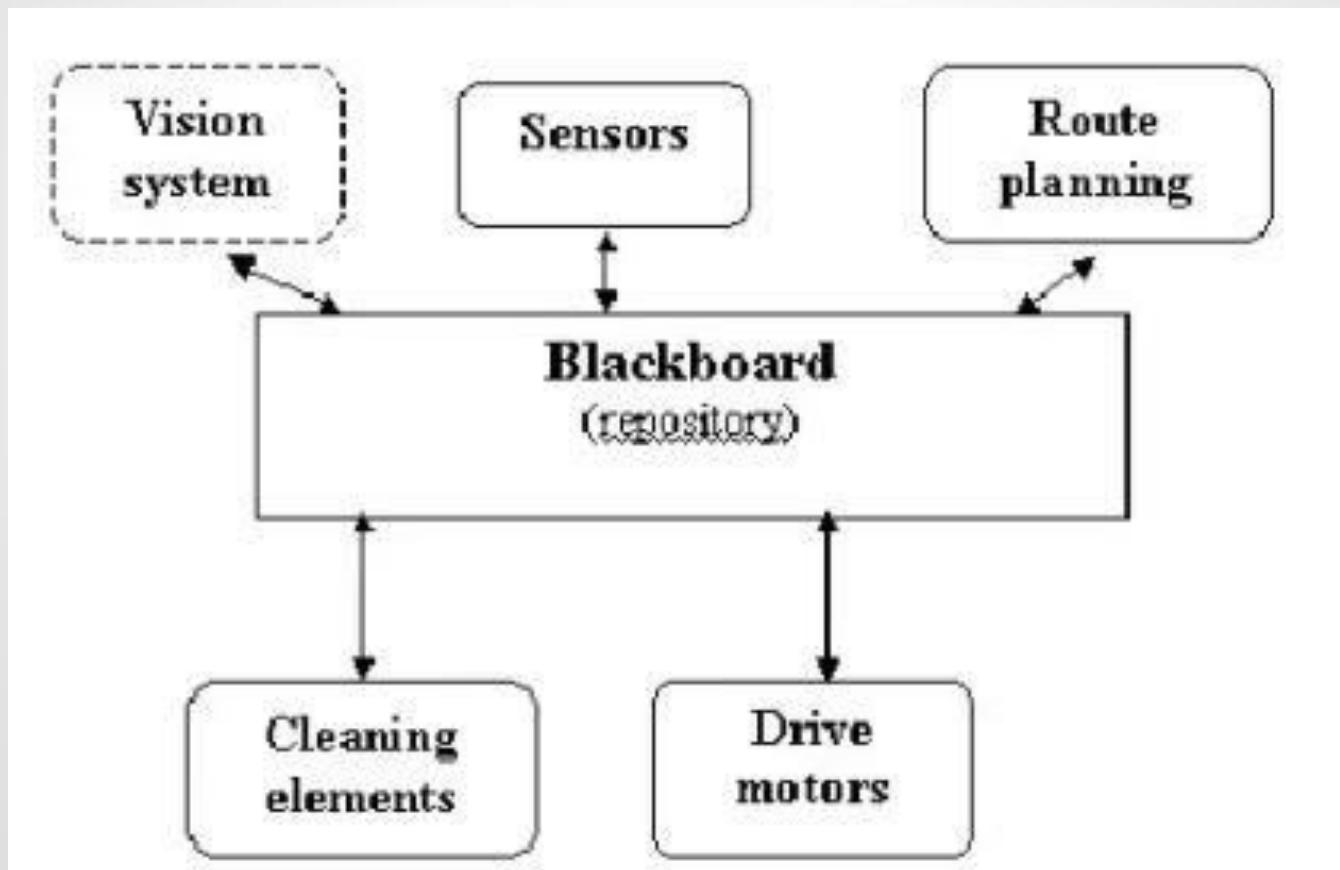
A repository architecture

Given reasons for your answer, suggest an appropriate structural model for the following systems:

- A **robot floor-cleaner** (standalone system) that is intended to clean relatively clear spaces such as corridors. The cleaner must be able to sense walls and other obstructions.

A repository architecture

- A robot floor-cleaner that is intended to clean relatively clear spaces such as corridors. The cleaner must be able to sense walls and other obstructions.



A repository architecture

- A robot floor-cleaner that is intended to clean relatively clear spaces such as corridors. The cleaner must be able to sense walls and other obstructions.
- The most appropriate model is a **repository model**, with each of the subsystems (wall and obstacle sensors, path planning, vision (perhaps), etc.) placing information in the repository for other subsystems to use.
- Robotic applications are in the realm of Artificial intelligence, and for the AI systems such as this, a special kind of repository called a **blackboard** (where the presence of data activates particular subsystems) is normally used

Plug-In Architecture

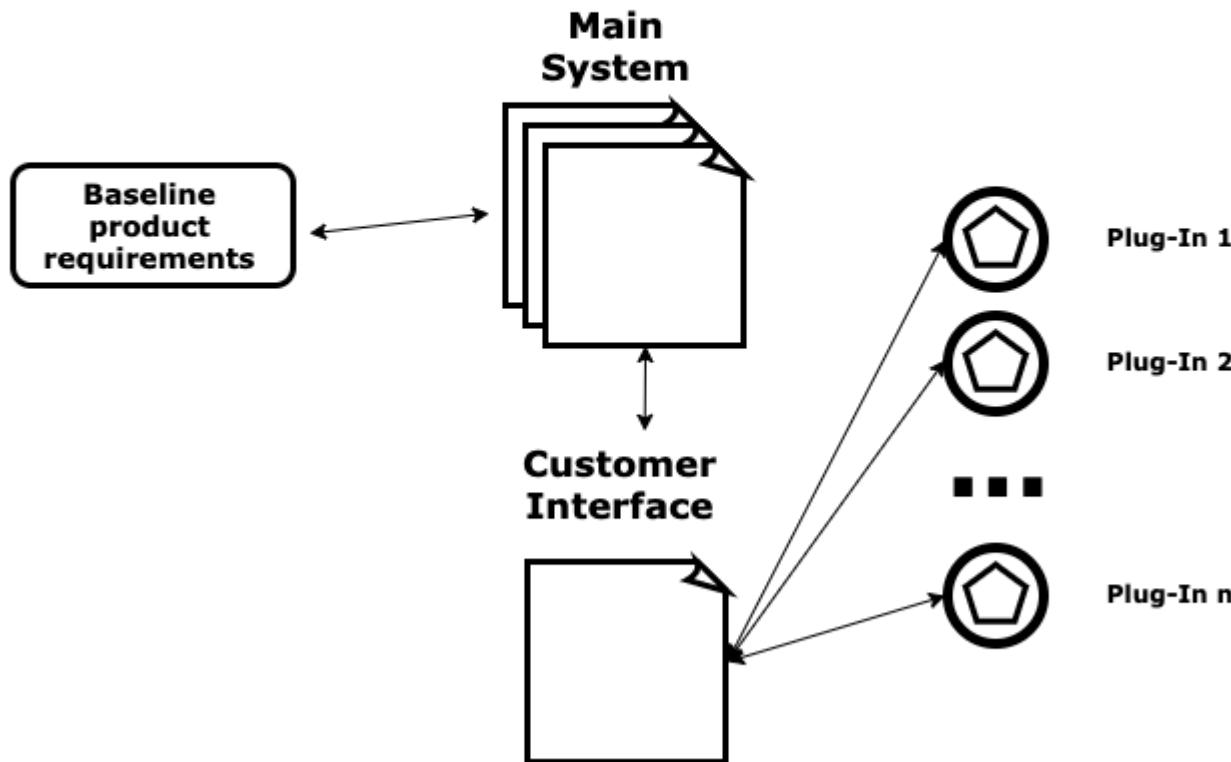
- ❖ Many people are worried about privacy, whether it's protecting their banking data, passwords, or making sure no one is reading their messages or emails. One way to ensure privacy protection is via encryption, and many software programs do so.
- ❖ Most encryption applications are installed in our browser (Chrome, Mozilla, Explorer, etc.) as an add-on: a module that works on top of the browser for any communication and encrypts it. This application is called a **plug-in**.

Plug-In Architecture

- ✧ A plugin architecture is an architecture that will **call external code at certain points without knowing all the details of that code in advance.**
- ✧ A plug-in is a **bundle** that **adds functionality** to an application, called the **host application**, through some well-defined architecture for extensibility. This **allows third-party developers** to add functionality to an application without having access to the source code.
- ✧ This also allows users to add new features to an application just by installing a new bundle in the appropriate folder.

Plug-In Architecture

Plug-In Architecture High Level Diagram



Plug-In Architecture

A standard plug-in architecture has four parts:

- **Baseline product requirements:** This is the **set of minimal requirements that define the application**, determined at the beginning of the development process when an initial set of features were included in the product.
- **Main system:** This is the application **we plug the plug-ins to**. The main system needs to provide a way to integrate plug-ins, and therefore will slightly vary the original baseline product to ensure compatibility.
- **Customer Interface:** This is the module that interacts with the customer, for example, a web browser (Chrome, Mozilla, etc.).
- **Plug-ins:** These are add-ons that enlarge the minimal requirements of the application and give it extra functionality.

Plug-In Architecture

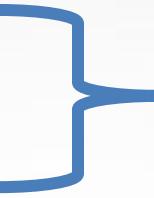
There are several **advantages** to using plug-in architecture:

- The plug-in architecture is the **best way to add particular functionality** to a system that was not initially designed to support it.
- This architecture removes limits on the amount of functionality an application can have. We can add infinite plug-ins (The Chrome browser has hundreds of plug-ins, called extensions).
- No rewriting the system.

There are also a few key **disadvantages**:

- Plug-ins **can be a source of viruses and attacks from external players**.
- Having many plug-ins in an application may affect its performance.
- Plug-ins frequently **crash with each other** and produce malfunctions in the main system.

Real-World Example

- SendSafely (sendsafely.com)
 - Mailvelope (mailvelope.com)
 - Rapportive (rapportive.com) connects LinkedIn to your browser.
 - Trello (trello.com) connects Trello to your browser.
- 
- Encryption software

Case-Study: Solving a Business Problem With Plug-In Architecture

Pacific is a retail website that is very successful in Southeast Asia. Here are some **quick facts**:

- The company sells more than 64,000 items on its website, mainly to Southeast Asian customers.
- About 12 items are sold every **second** on the website, 24 hours a day, 365 days a year.
- There are heavy users of the site: users that buy many items at once, and they need to do it quickly.

What's the Business Problem?

Each time a heavy user wants to buy many items at once, he or she must **open separate browser windows** for each item, causing confusion and frustration, which can stop the user from making a purchase.

For example, some users select ten items, but in the end they buy only six because they go back and forth from the shopping cart to the item pages.

How can this problem be solved?

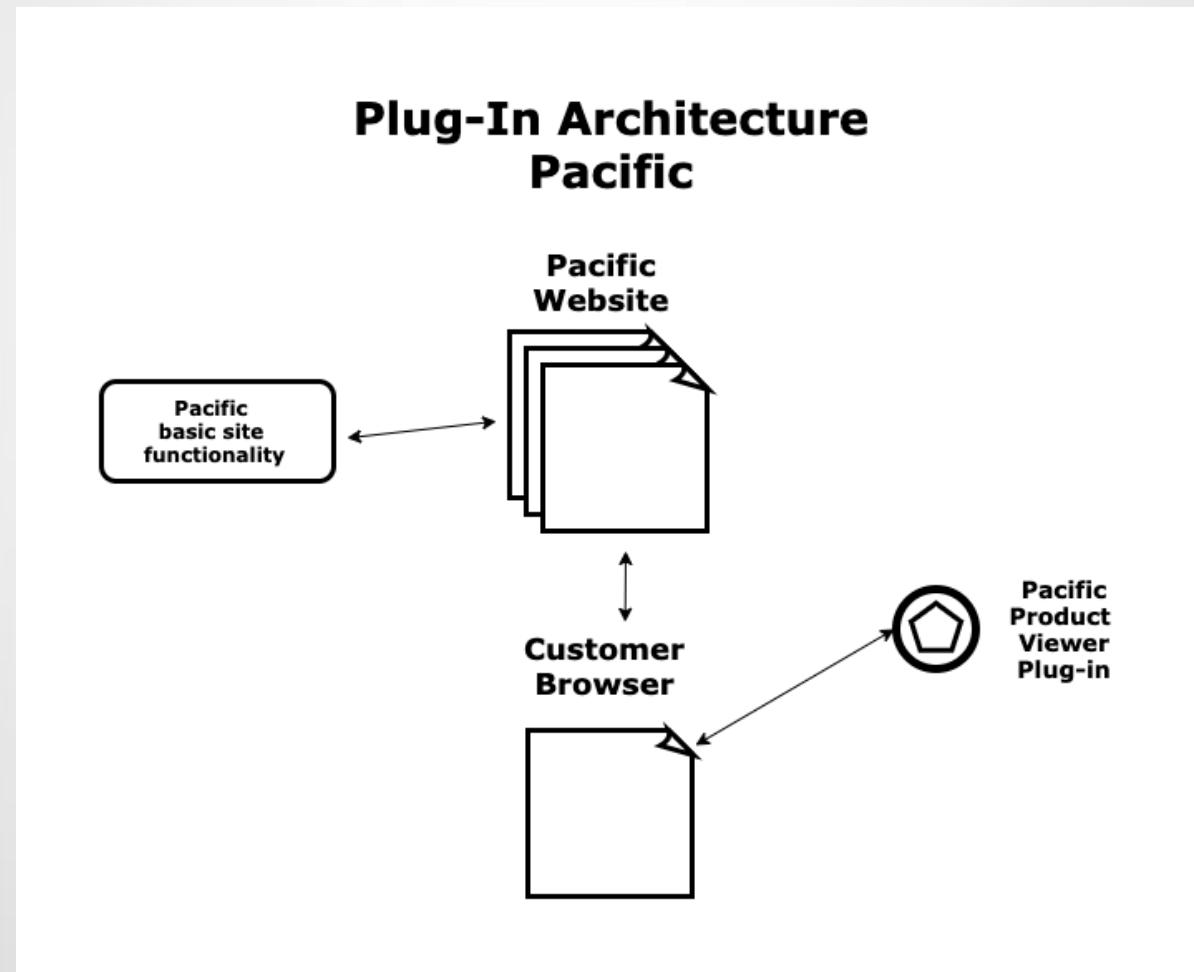
Pacific Product Viewer, a plug-in for any browser

Pacific has developed a plug-in for Chrome called Pacific Product Viewer: when the user installs the plug-in, a list of interesting items is shown in a pop-up window, and there is no need to open multiple tabs.

This window allows the user to see many items without having to switch tabs. It is usually located in the upper-right corner of the screen.

Case-Study: Solving a Business Problem With Plug-In Architecture

- **Pacific website:** It is the main Pacific website, developed according to a minimal set of requirements.
- **Customer browser:** Chrome, Mozilla, Explorer, etc.
- **Pacific Product Viewer Plug-in:** The piece of software that was developed as an add-on for extra functionality.



Distributed Systems Architectures

❖ **Distributed Systems Architectures**

- ❖ Client-Server Architecture
- ❖ Broker Architectural Style : CORBA
- ❖ Service-Oriented Architecture (SOA)

Distributed Architecture

- A distributed system is "**a collection of independent computers that appears to the user as a single coherent system.**" Information processing is distributed over several computers rather than confined to a single machine.
- Virtually all large computer-based systems are now distributed systems.
- **Distributed software engineering** is therefore very important for enterprise computing systems.

Distributed Architecture

- A distributed system can be demonstrated by the client-server architecture which forms the base for multi-tier architectures; alternatives are the broker architecture such as CORBA, and the Service-Oriented Architecture (SOA).
- There are several technology frameworks to support distributed architectures, including .NET, J2EE, CORBA, .NET Web services, AXIS Java Web services, and Globus Grid services.

Distributed Architecture

- **Middleware** is an infrastructure that appropriately supports the development and execution of distributed applications.
- It provides a **buffer** between the **applications and the network**.
- It sits in the middle of system and **manages or supports the different components** of a distributed system.
- Examples are transaction processing monitors, data convertors and communication controllers etc.

Distributed System

Application 1

Application 2

Application 3

Middleware platform

Personal Computer

Personal digital
assistance

Mainframe

Middleware as an infrastructure for distributed system

Distributed System

Advantages

- **Resource sharing** – Sharing of hardware and software resources.
- **Openness** – Flexibility of using hardware and software of different vendors.
- **Concurrency** – Concurrent processing to enhance performance.
- **Scalability** – Increased throughput by adding new resources.
- **Fault tolerance** – The ability to continue in operation after a fault has occurred.

Disadvantages

- **Complexity** – They are more complex than centralized systems.
- **Security** – More susceptible to external attack.
- **Manageability** – More effort required for system management.
- **Unpredictability** – Unpredictable responses depending on the system organization and network load.

Centralized System vs. Distributed System

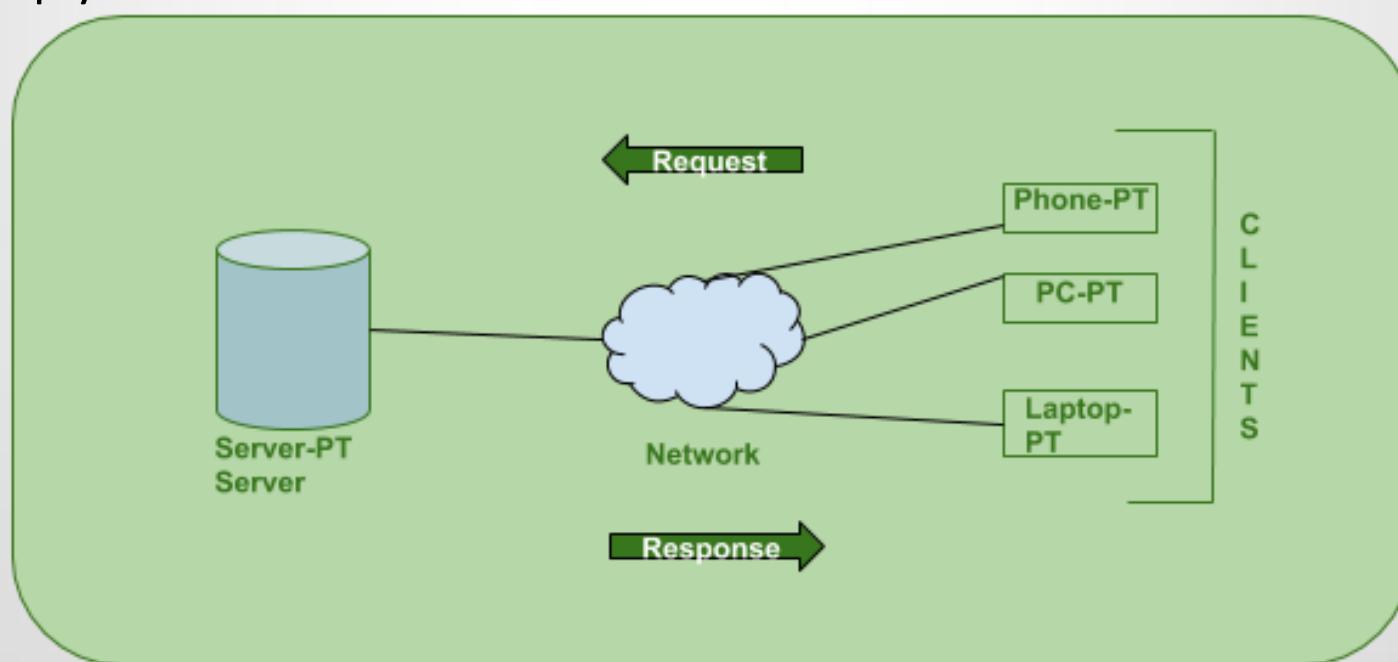
Criteria	Centralized system	Distributed System
Economics	Low	High
Availability	Low	High
Complexity	Low	High
Consistency	Simple	High
Scalability	Poor	Good
Technology	Homogeneous	Heterogeneous
Security	High	Low

Client-Server Architecture

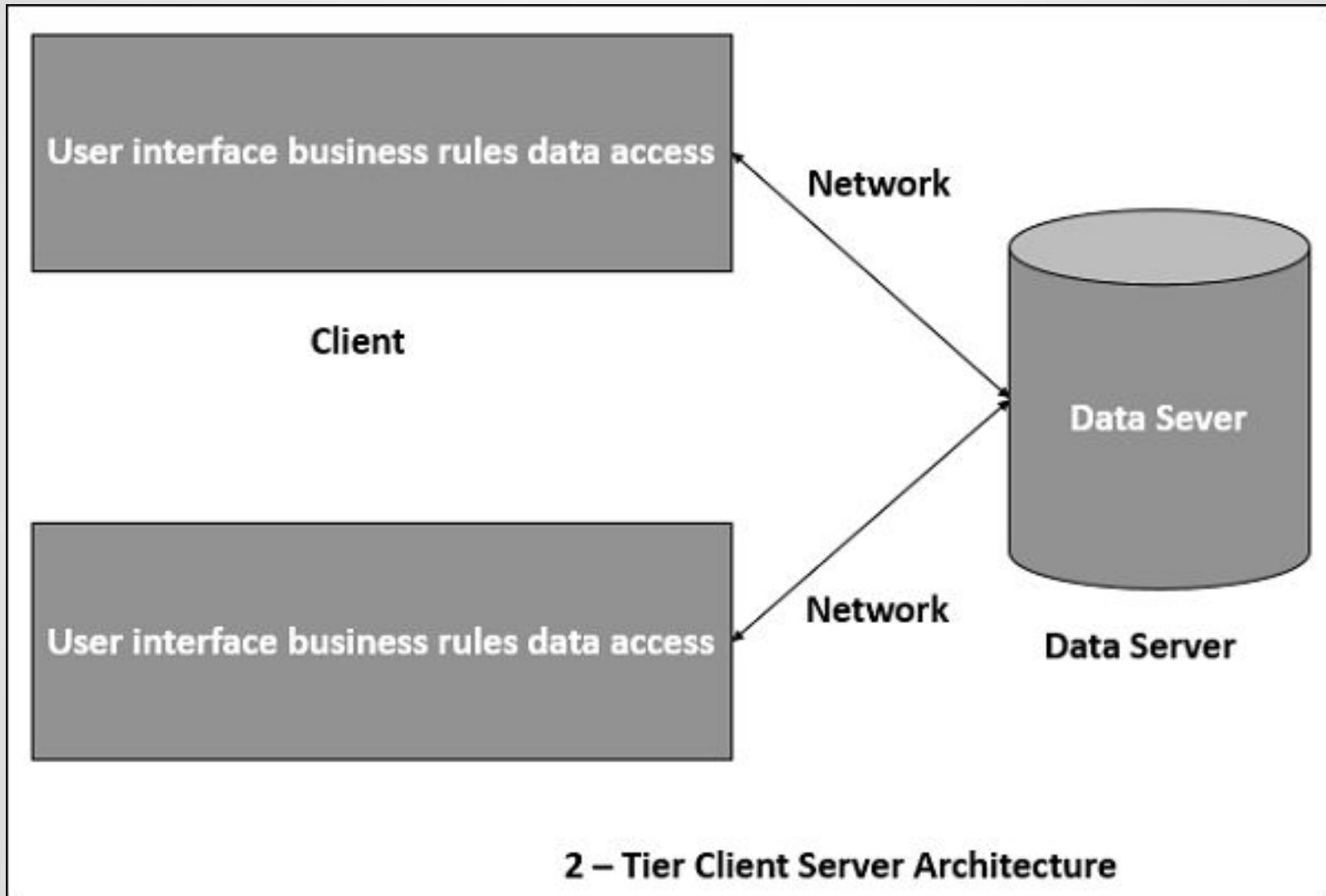
The client-server architecture is the most common distributed system architecture which decomposes the system into two major subsystems or logical processes –

Client – This is the first process that issues a **request to** the second process i.e. the server.

Server – This is the second process that **receives the request**, carries it out, and sends a reply to the client.



Client-Server Architecture



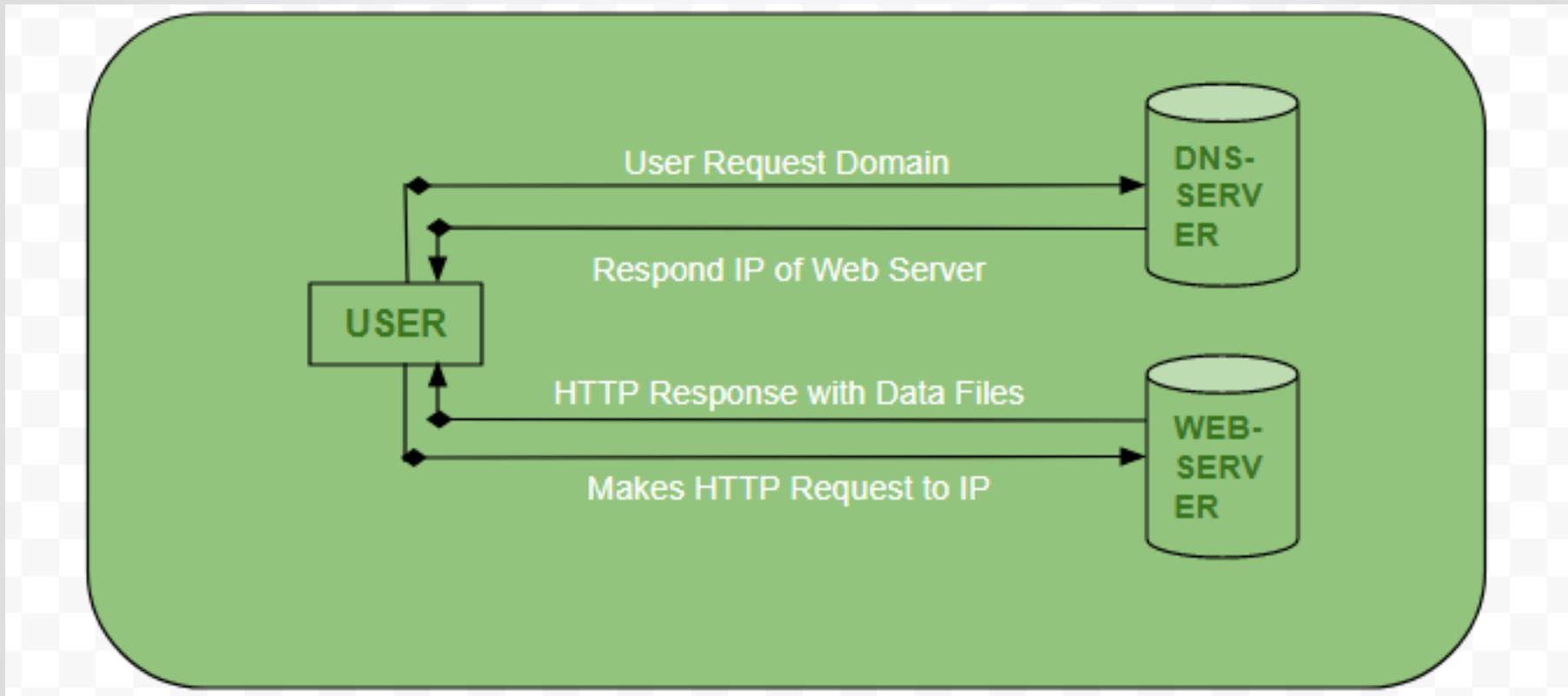
Client-Server Architecture

How the Client-Server Model (web browsers) works ?

- User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.
- **DNS Server** lookup for the address of the **WEB Server**.
- **DNS Server** responds with the **IP address** of the **WEB Server**.
- Browser sends over an **HTTP/HTTPS** request to **WEB Server's IP** (provided by **DNS server**).
- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.
-

Client-Server Architecture

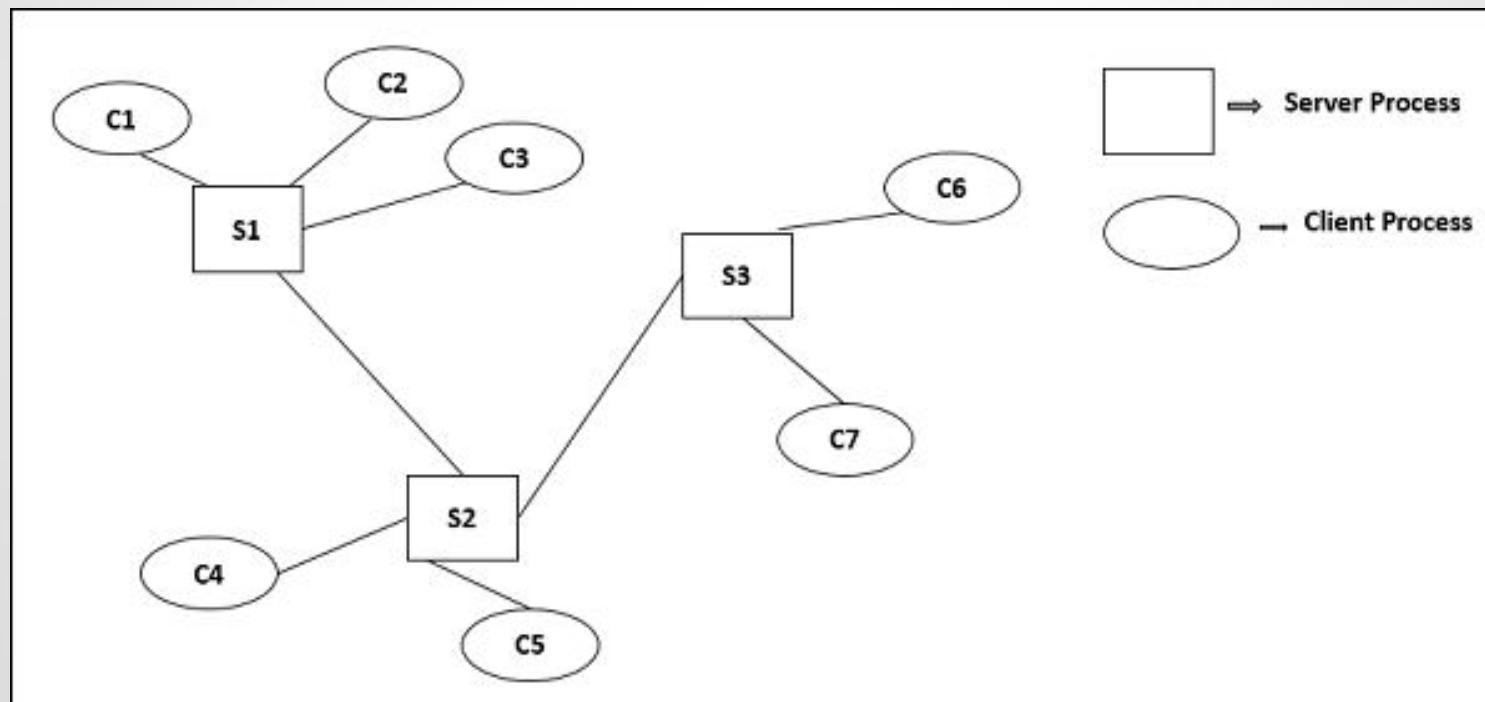
How the Client-Server Model (web browsers) works ?



Multi-Tier Architecture (n-tier Architecture)

Multi-tier architecture is a client–server architecture in which the functions such as **presentation**, **application** processing (Business Logic, Logic Tier, or Middle Tier), and **data** management are physically separated.

** 3-Tier Architecture, 2-Tier Architecture



Client-Server Architecture

Advantages

- Distribution of data is straightforward;
- Makes effective use of networked systems. May require cheaper hardware;
- Easy to add new servers or upgrade existing servers.

Disadvantages

- No shared data model so sub-systems use different data organisation. Data interchange may be inefficient;
- Redundant management in each server;
- No central register of names and services - it may be hard to find out what servers and services are available.

Client-Server Architecture

Here are some real-life situations that this would be useful for:

- Enterprise resource planning (ERP) system
- SAP (sap.com).
- Oracle Business Suite (oracle.com).
- Microsoft Dynamics (microsoft.com).
- Infor (infor.com).
- Epicor (epicor.com).

Case-Study: Solving a Business Problem With Client-Server Architecture

IrisGold is a gold mining company. Here are some **quick facts**:

- IrisGold operates on **three continents**, with more than **21,000 employees**.
- The company's mines are mostly **located in remote places** like the Amazonas in Brazil, the Andes mountain range, the Ural mountains in Russia, and eastern South Africa.
- The company is selecting an Enterprise Resource Planning (ERP) system package. How does this inform your decision?

What's the Business Problem?

IrisGold wants to deploy and operate its new ERP package securely. But they have **two main constraints**:

- Its users are in **remote places** in the world with different kinds of devices (laptops, notebooks, phones, tablets).
- **Client devices must be light:** they must be a simple notebook computer, tablet, or phone with few processing and storage capabilities, **able to communicate to a remote server.**

Case-Study: Solving a Business Problem With Client-Server Architecture

Let's start with the facts!

1. We know that users are scattered over the world and use different devices that need to be lightweight.
2. We know that users have deficient infrastructure capabilities and work in remote places.
3. Clients need only to be able to connect with a central server.
4. The ERP system installed in the central server must cover all the business rules. In essence, it must manage all modules for all countries internally, and it must answer client requests by communicating with a central database.

So to sum up:

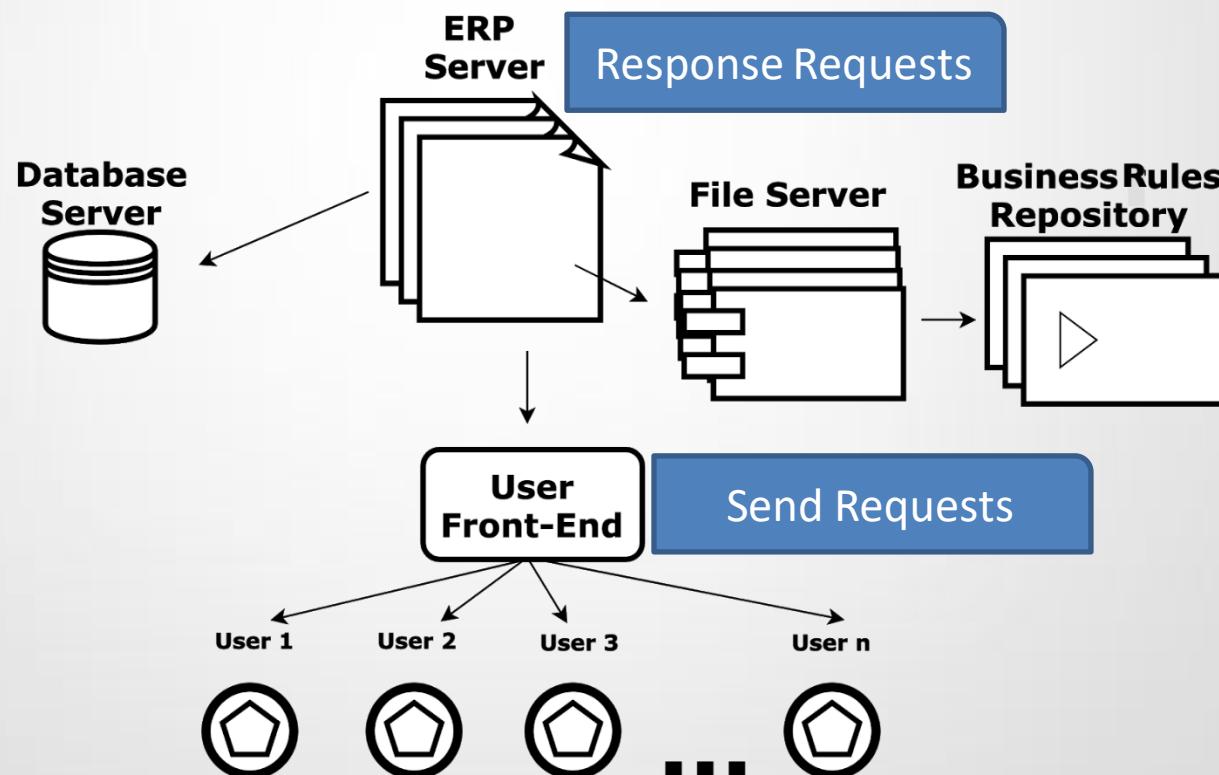
1. We know that users are scattered over the world and use different devices that need to be lightweight.
2. We know that users have deficient infrastructure capabilities and work in remote places.
3. Clients need only to be able to connect with a central server.
4. The ERP system installed in the central server must cover all the business rules. In essence, it must manage all modules for all countries internally, and it must answer client requests by communicating with a central database.

Case-Study: Solving a Business Problem With Client-Server Architecture

What's the Solution?

All heavy processing is done at IrisGold's headquarters. Clients are thin; that's why they are called "thin clients." They do not process or store large amounts of data. They just speak with the server.

Client-Server Architecture IrisGold



Case-Study: Solving a Business Problem With Client-Server Architecture

Front-End: This is the piece of software that interacts with ERP users, even if they are in different countries.

ERP server: This is the server where the ERP software is installed.

File server: The ERP server requests files from the file server to fulfill user requests. Examples: an invoice printed in PDF format, a report, a data file that the user needs. It is good practice to install a file server when the application has many users that read, update, or write files frequently.

Business rules repository: A repository of business procedures, methods, and regulations for every country (all different), to make the ERP work according to each country's needs. This repository is often separated from the software to make customizations more agile and secure. It is a good practice introduced by ERPs in the '90s that has spread over many kinds of non-ERP applications.

Database server: This server contains the tables, indexes, and data managed by the ERP system. Examples: customer table, provider table, invoice list, stock tables, product IDs, etc.

Broker Architectural Style



Broker Architectural Style

Broker Architectural Style is a middleware architecture used in distributed computing to **coordinate and enable** the communication between **registered servers and clients**. Here, object communication takes place through a middleware system called an **object request broker(ORB)**.

- Client and the server **do not interact** with each other **directly**. Client and server have a direct connection to **its proxy** which communicates with the mediator-broker.
- A server provides services by registering and publishing their interfaces with the broker and clients can request the services from the broker statically or dynamically by look-up.
- **CORBA** (Common Object Request Broker Architecture) is a good implementation example of the broker architecture.

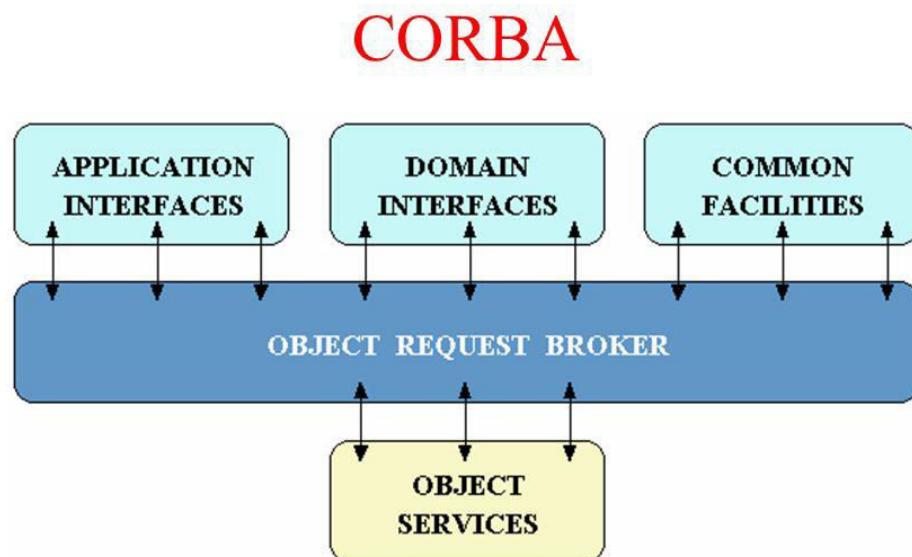
Broker implementation in CORBA

CORBA, or **Common Object Request Broker Architecture**, is a standard architecture for **distributed object systems**. It allows a distributed, heterogeneous **collection of objects** to interoperate.

CORBA is a standard **defined by the OMG** (Object Management Group). It describes an architecture, interfaces, and protocols that distributed objects can use to interact with each other.

Part of the CORBA standard is the **Interface Definition Language (IDL)**, which is an implementation-independent language for **describing the interfaces of remote objects**.

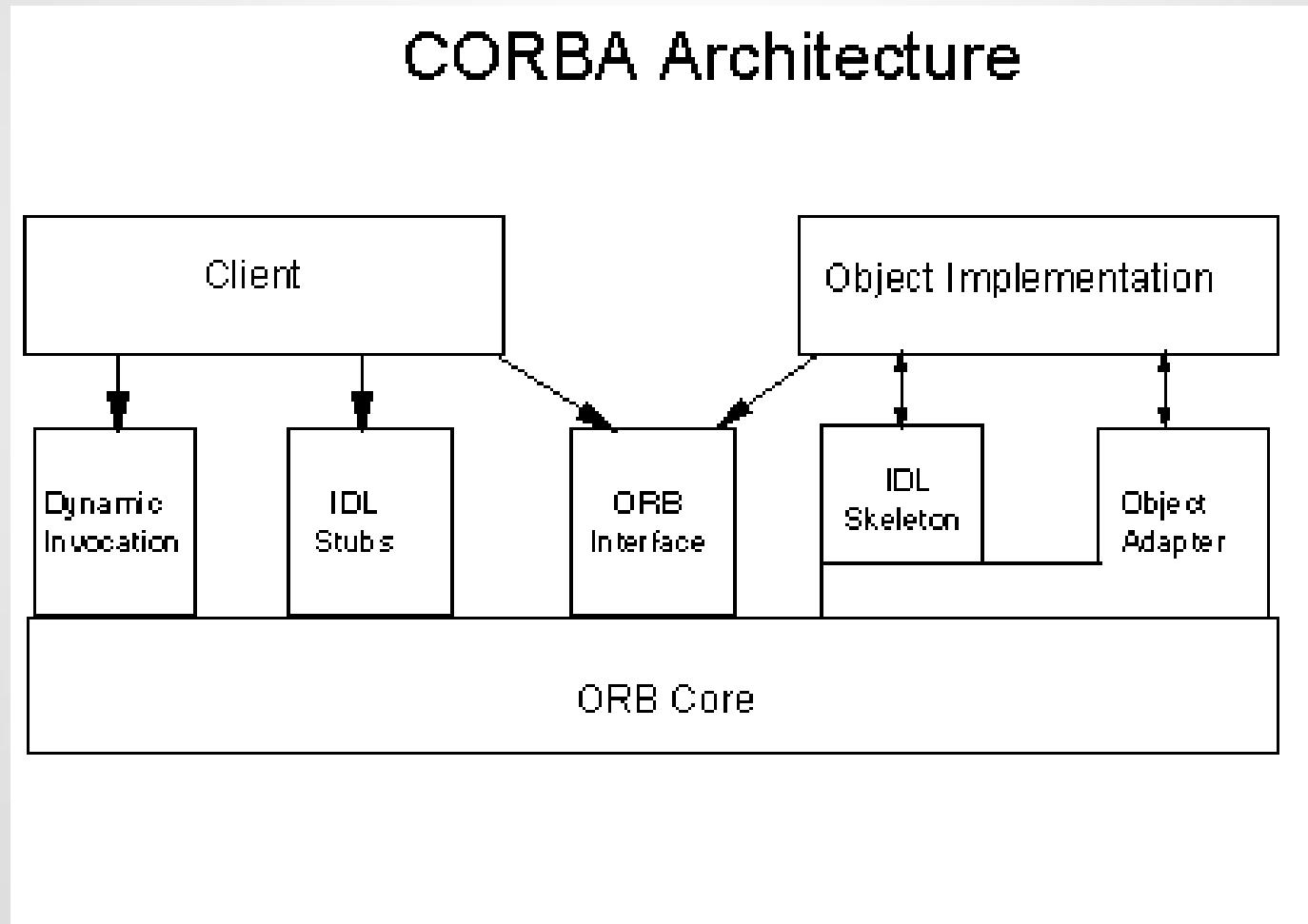
CORBA **describes a messaging mechanism** by which objects distributed over a network can communicate with each other irrespective of the platform and language used to develop those objects.



OMG Reference Model architecture

Broker implementation in CORBA

The ORB manages the interactions between clients and object implementations (Server).



Broker implementation in CORBA

Client Side Architecture

The client side architecture provides clients with interfaces to the ORB and object implementations. It consists of the following interfaces:

Dynamic Invocation - This interface allows for the **specification of requests at runtime**. This is necessary when **object interface is not known at run-time**. Dynamic Invocation **works in conjunction with the interface repository**.

IDL Stub - This component consists of **functions generated by the IDL interface definitions** and linked into the program. The functions are a **mapping between the client and the ORB implementation**. Therefore ORB capabilities can be made available for any client implementation for which there is a language mapping. Functions are called just as if it was a **local object**.

ORB Interface - The ORB interface may be called by either the client or the object implementation. The interface **provides functions of the ORB** which may be directly accessed by the client (such as retrieving a reference to an object.) or by the object implementations. This interface is **mapped to the host programming language**. The ORB interface must be supported by any ORB.

ORB core - Underlying mechanism used as **the transport level**. It provides **basic communication of requests to other sub-components**.

Broker implementation in CORBA

Implementation Side Architecture

The implementation side interface consists of the ORB Interface, ORB core and IDL Skeleton Interface defined below:

IDL Skeleton Interface - The ORB calls method **skeletons** to **invoke the methods** that were requested from clients. Object Adapters (OA) - Provide the means by which object implementations **access most ORB services**. This includes the generation and interpretation of **object references, method invocation, security and activation**. The object adapter actually exports three different interfaces: a private interface to skeletons, a private interface to the ORB core and a public interface used by implementations. The OA **isolates the object implementation from the ORB core**.

Requests

The client **requests a service** from the object implementation. The **ORB transports the request**, which invokes the method using object adapters and the IDL skeleton.

The client has an object reference, an operation name and a set of parameters for the object and activates operations on this object. The Object Management Group / Object Model defines each operation to be associated with a controlling parameter, implemented in CORBA as an object reference. **The client does not know the location of the object or any of the implementation details**. The request is handled by the ORB, which must locate the target object and route the request to that object. It is also responsible for getting results back to the client.

Broker implementation in CORBA

Object Adapters

Object Adapters (OA) are the primary ORB service providers to object implementations. OA have a public interface which is used by the object implementation and a private interface that is used by the IDL skeleton.

Example services provided by OA's are:

- Method invocation (in conjunction with skeleton),
- Object implementation activation and deactivation,
- Mapping of object reference to object implementations,
- Generation of object references, and
- Registering object implementations, used in locating object implementations when a request comes in.

ORB Interface Operations

The ORB provides operations on references and strings. Two operations for converting an object reference to strings and back again. An is_nil operation for testing whether an object reference is referencing no object. A duplicate operation allows for a duplicate reference to the same object to be created. A release operation is provided to reclaim the memory used by an object reference.

Broker implementation in CORBA

Object Services

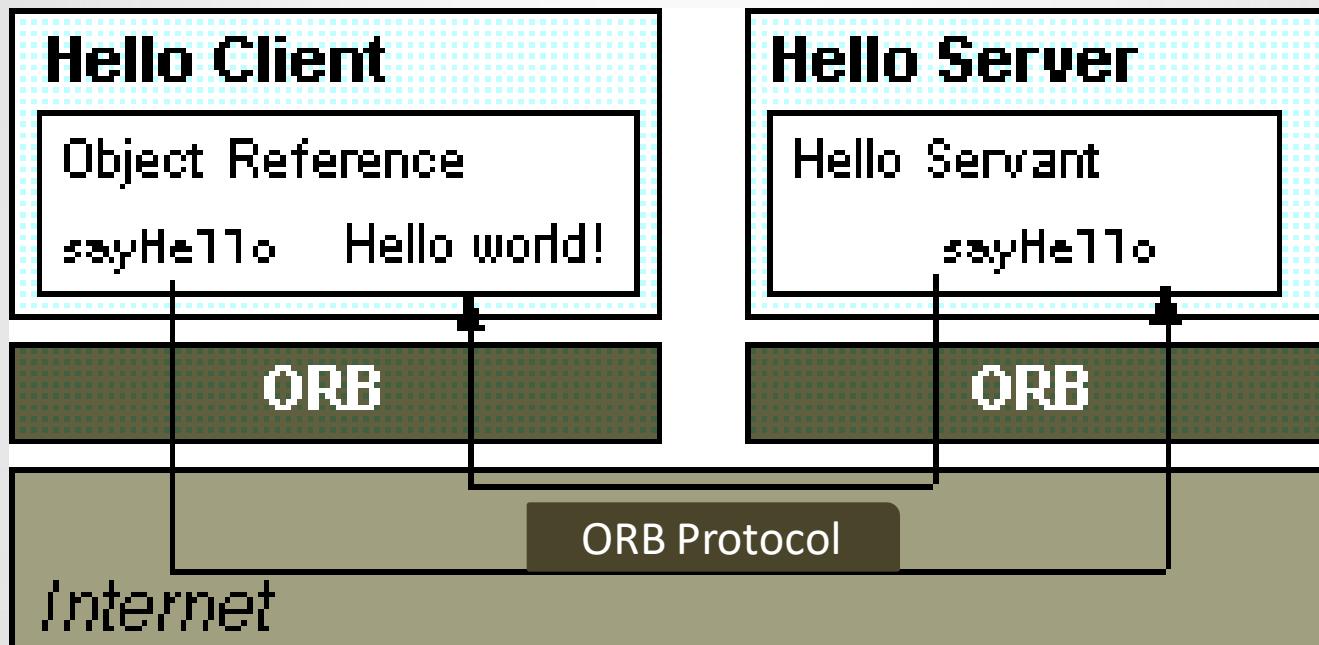
Object services refer to fundamental services provided by their own objects that support common interactions between other objects. The services follow the OMG Common Object Services Specification (COSS) guidelines.

Current object services include:

- **Event Management services** which refer to the asynchronous communication of different CORBA objects with one another.
- **Naming Object services** that maps a human-readable name (string) to an object relative to its context.
- **Persistent services** assure that an object (CORBA developed) outlives its creator. It allows an objects state to be retained at a given point in time. This feature is used when a host crashes or is rebooted.
- **Externalization services** collect objects and transport them as serial objects.
- **Life-cycle services** determine the methods for an object creation and termination.
- **Concurrency services** provide for distributed locks on a given object.
- **Relationship Objects** are used for CORBA object modeling and graphing.
- **Transaction object services** allow the sharing of a transaction among multiple objects.

Broker implementation in CORBA

Describe this architecture based on ORB style and mention its all objects services?



Service-Oriented Architecture

Cloud computing

The term **Cloud** refers to a **Network or Internet**. In other words, we can say that Cloud is something, which is present at **remote location**. Cloud can provide services over public and private networks, WAN, LAN or VPN.

Cloud computing is the **on-demand availability** of computer system resources, especially data storage and computing power, **without direct active management** by the user.

Cloud computing (also called simply, the cloud) describes the act of storing, managing and processing data online - as opposed to on your own physical computer or network.

Cloud computing is the delivery of **different services through the Internet**. These resources include tools and applications like data storage, servers, databases, networking, and software.

Cloud computing is the delivery of **on-demand computing services over the internet** on a **pay-as-you-go basis**.

Cloud computing

Types of Cloud Computing

Public Cloud: Multi-tenant environment with pay-as-you-grow scalability

A public cloud is built over the Internet and can be **accessed by any user who has paid for the service.** Public clouds are owned by service providers and are accessible through a subscription.

Many public clouds are available, including **Google App Engine (GAE)**, **Amazon Web Services (AWS)**, **Microsoft Azure**, IBM Blue Cloud, and Salesforce.com's Force.com.

Private Cloud: Scalability plus the enhanced security and control of a single-tenant environment

A private cloud is built within the domain of an intranet **owned by a single organization.** Therefore, it is client owned and managed, and its access is limited to the owning clients and their partners. Its deployment was not meant to sell capacity over the Internet through publicly accessible interfaces.

Cloud computing: Cost Model

Types of Cloud Computing

Hybrid Cloud: Connect the public cloud to your private cloud or dedicated servers - even in your own data center.

Private clouds can also support a hybrid cloud model by supplementing local infrastructure with computing capacity from an external public cloud.

For example, the Research Compute Cloud (RC2) is a private cloud, built by IBM, that interconnects the computing and IT resources at eight IBM Research Centers scattered throughout the United States, Europe, and Asia. A hybrid cloud provides access to clients, the partner network, and third parties.

Cloud computing

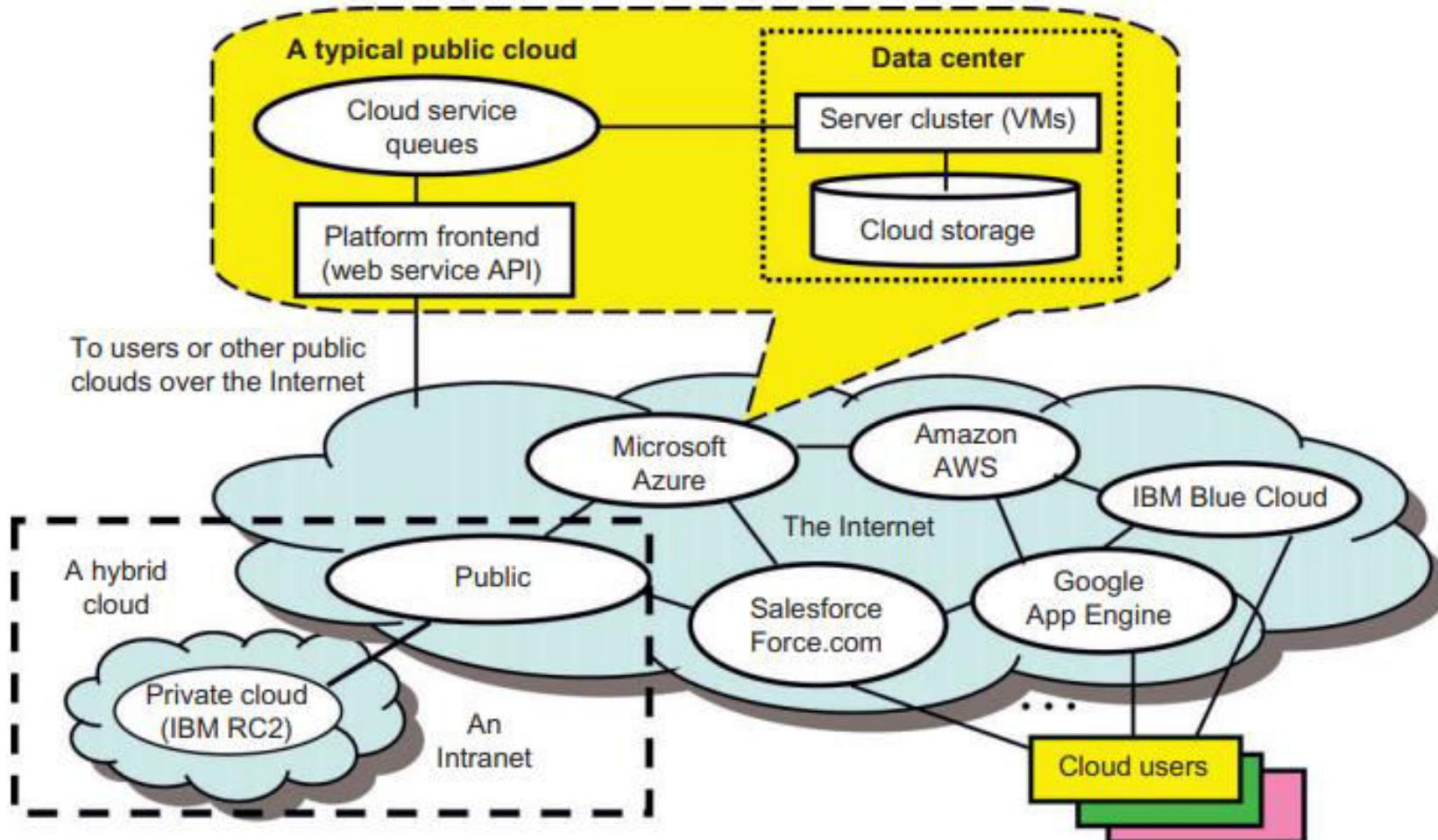
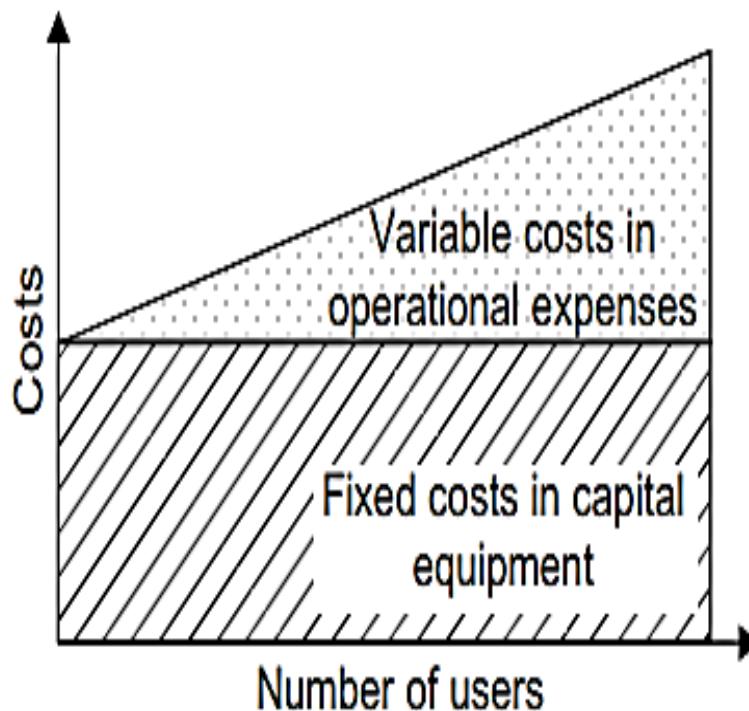


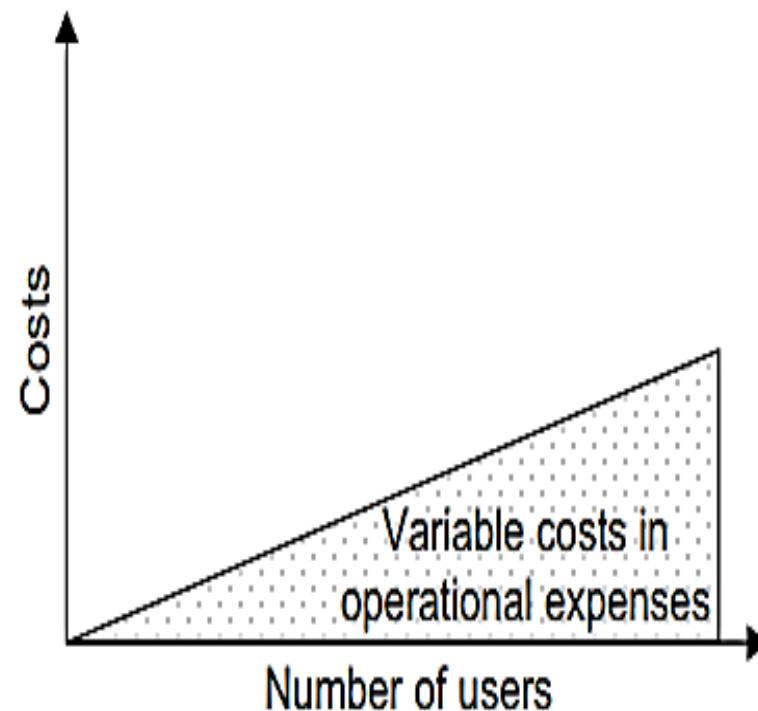
FIGURE 4.1

Public, private, and hybrid clouds illustrated by functional architecture and connectivity of representative clouds available by 2011.

Cloud computing : Cost



(a) Traditional IT cost model



(b) Cloud computing cost model

FIGURE 4.3

Computing economics between traditional IT users and cloud users.

Service Models

Cloud computing is based on service models. These are categorized into three basic service models which are –

- ❖ Infrastructure-as-a-Service (IaaS)
- ❖ Platform-as-a-Service (PaaS)
- ❖ Software-as-a-Service (SaaS)

Cloud computing

Infrastructure as a service (IaaS)

- Most basic cloud service model
- Cloud providers offer computers, as **physical** or more often as virtual machines, and other resources. This model allows users to use **virtualized IT resources** for computing, storage, and networking.
- Virtual machines are run as guests by a hypervisor, such as Xen or KVM.
- Cloud users deploy their applications by then installing operating system images on the machines as well as their application software.
- Cloud providers typically bill IaaS services on a utility computing basis, that is, **cost will reflect the amount of resources allocated and consumed**.

Examples of IaaS include: Amazon Cloud Formation (and underlying services such as Amazon **EC2**), Rackspace Cloud, Terremark, and **Google Compute Engine**

***A hypervisor is computer software, firmware or hardware that creates and runs virtual machines*

Cloud computing

Platform as a service (PaaS)

- Cloud providers deliver **a computing platform** typically including operating system, programming language execution environment, database, and web server.
- Application developers **develop and run their software** on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers.
- Examples of PaaS include: Amazon Elastic Beanstalk, Cloud Foundry, Heroku, Force.com, EngineYard, Mendix, **Google App Engine**, Microsoft Azure and OrangeScape.

Software as a service (SaaS)

- Cloud providers install and operate application software in the cloud and **cloud users access the software from cloud clients.**
- The pricing model for SaaS applications is typically a monthly or yearly flat fee per user, so price is scalable and adjustable if users are added or removed at any point.
- **Examples of SaaS include:** Google Apps, innkeypos, Quickbooks Online, Limelight Video Platform, Salesforce.com, and Microsoft Office 365.

Cloud computing Services

Cloud Services

Traditional
On-Premise
(On-Prem)

Applications
Data
Runtime
Middleware
Operating System
Virtualization
Servers
Storage
Networking

Infrastructure
as a Service
(IaaS)

Applications
Data
Runtime
Middleware
Operating System
Virtualization
Servers
Storage
Networking

Platform
as a Service
(PaaS)

Applications
Data
Runtime
Middleware
Operating System
Virtualization
Servers
Storage
Networking

Software
as a Service
(SaaS)

Applications
Data
Runtime
Middleware
Operating System
Virtualization
Servers
Storage
Networking

Self Managed

Managed By Vendor

Service-Oriented Architecture (SOA)

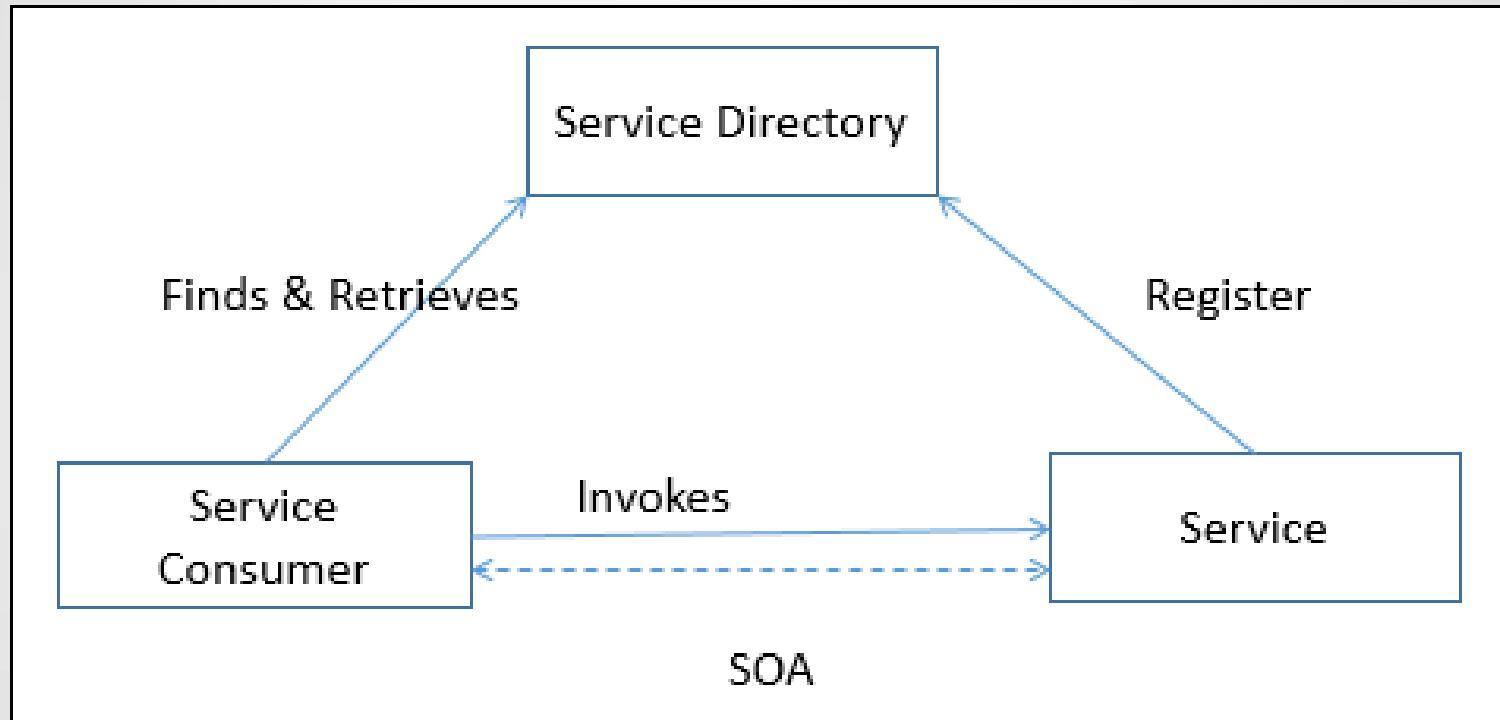
Service-Oriented Architecture (SOA) is a **style of software design** where **services** are provided to the other components by application components, through a communication protocol over a network.

There are two major roles within Service-oriented Architecture:

Service provider: The service provider is **the maintainer of the service** and the organization that makes available one or more services for others to use. To advertise services, the provider can publish them in a registry, together with a service contract that specifies the nature of the service, how to use it, the requirements for the service, and the fees charged.

Service consumer: The service consumer can locate the service metadata in the registry and develop the required client components to bind and use the service.

Service-Oriented Architecture (SOA)



Service-Oriented Architecture (SOA)

Why to use SOA?

- SOA is widely used in market which responds quickly and makes effective changes according to market situations.
- The SOA keep **secret the implementation details** of the subsystems.
- It allows interaction of new channels with customers, partners and suppliers.
- It authorizes the companies to select software or hardware of their choice as it acts as platform independence.

Features

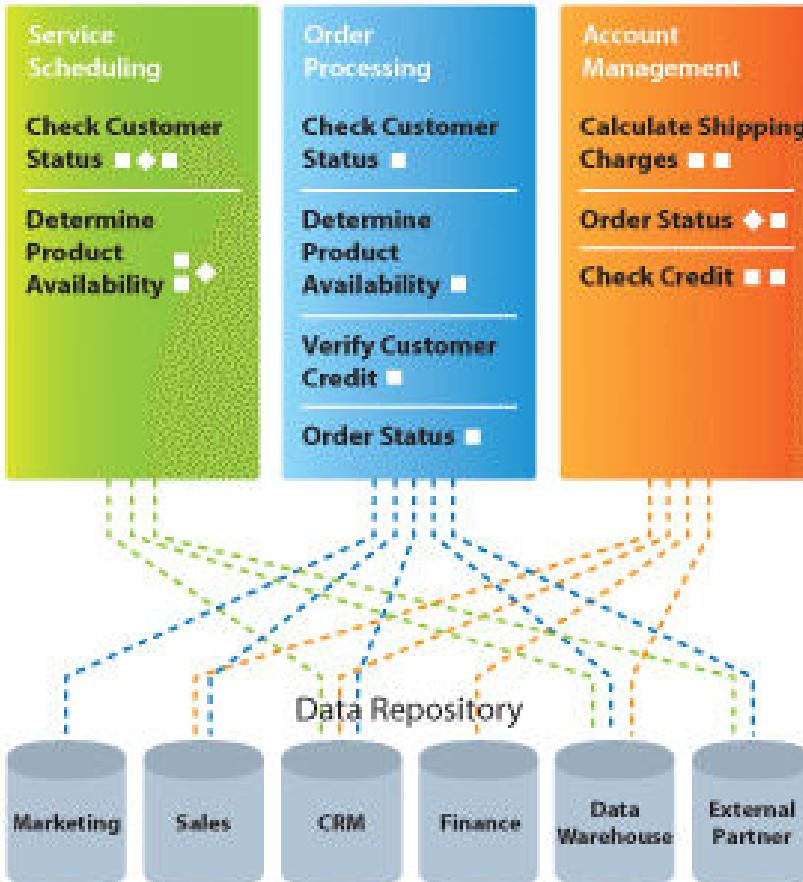
- SOA uses interfaces which **solves the difficult integration** problems in large systems.
- SOA communicates customers, providers and suppliers with messages by using the **XML schema**.
- It uses the message monitoring to improve the performance measurement and detects the security attacks.
- As it reuses the service, there will be lower software development and management costs.

Service-Oriented Architecture (SOA)

Before SOA

Closed - Monolithic - Brittle

Application Dependent Business Functions



After SOA

Shared services - Collaborative - Interoperable - Integrated

Composite Applications



Reusable Business Services

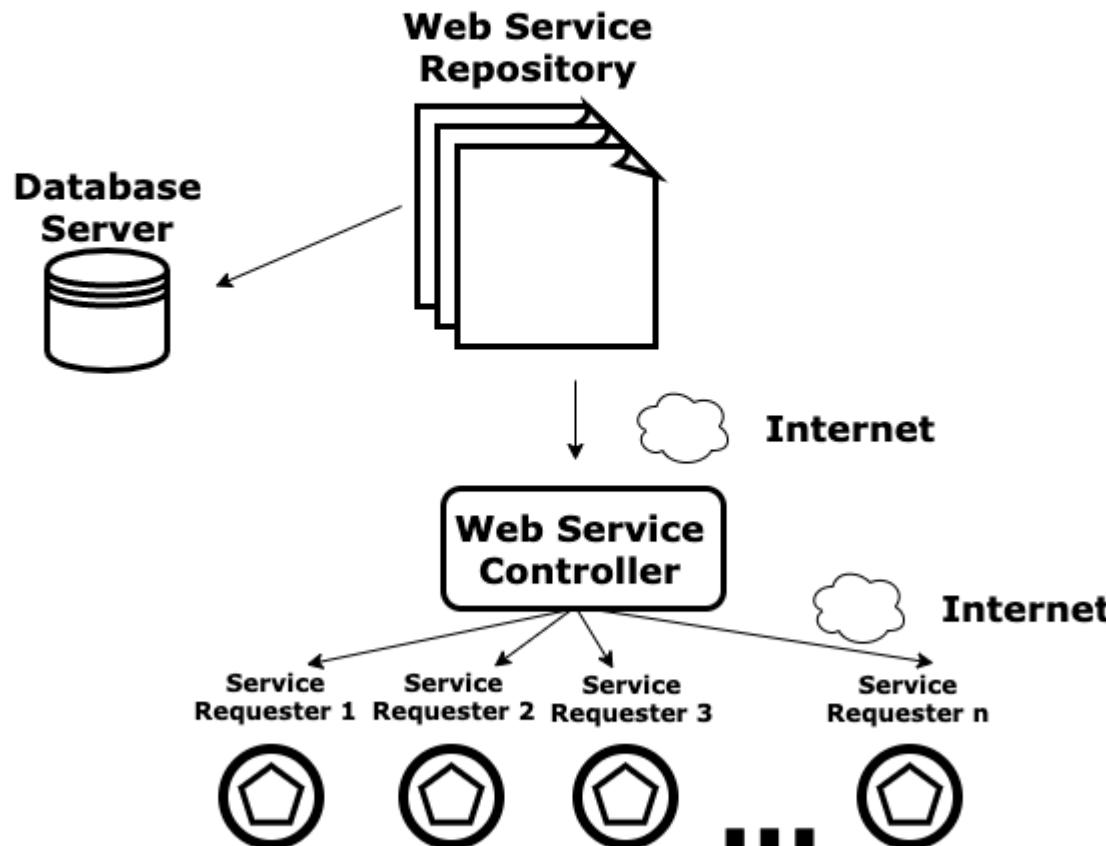


Data Repository



Service-Oriented Architecture (SOA)

Service Oriented Architecture High Level Diagram



Service-Oriented Architecture (SOA)

A standard client-server architecture has these parts:

Web service repository: This is a **library of web services** built to **serve external requests for information**. The served information is usually a little piece of information, like a number, a word, some variables, etc.

Web service controller: This module communicates the information in the web service repository with the **service requesters**. When an **external service requester calls** a certain function from the web service repository, the web service controller **interprets the call** and looks for the function in the web server repository. Then it executes the function and **returns a value to the requester**.

Service-Oriented Architecture (SOA)

Database server: This server contains the tables, indexes, and data managed by the core application. Searches and insert/delete/update operations are executed here.

Service requesters: These are **external applications that request services** from the web service repository through the internet, such as an organization requesting flight information from an airline or another company asking the package carrier for the location of a package at a given moment.

Note that service-oriented architecture is **created by the companies providing the service** - not the ones consuming it, and this is done to simplify the connections to possible clients.

Service-Oriented Architecture (SOA)

Advantages of SOA:

- **Service reusability:** In SOA, applications are made from existing services. Thus, services can be reused to make many applications.
- **Easy maintenance:** As services are independent of each other they can be updated and modified easily without affecting other services.
- **Platform independent:** SOA allows making a complex application by combining services picked from different sources, independent of the platform.
- **Availability:** SOA facilities are easily available to anyone on request.
- **Reliability:** SOA applications are more reliable because it is easy to debug small services rather than huge codes
- **Scalability:** Services can run on different servers within an environment, this increases scalability

Service-Oriented Architecture (SOA)

Disadvantages of SOA:

- **High overhead:** A validation of input parameters of services is done whenever services interact this **decreases performance** as it increases load and response time.
- **High investment:** A huge initial investment is required for SOA.
- **Complex service management:** When services interact they exchange messages to tasks. the number of messages may go in millions. It becomes a cumbersome task to handle a large number of messages.

Case-study: Solving a Business Problem With Service- Oriented Architecture

GlobalWeather is a global company that sells information about weather conditions. Here are some important facts:

- GlobalWeather is based in the U.S. and has more than 700 employees.
- GlobalWeather sells weather information to companies about countries, regions, cities, or even precise locations defined by latitude and longitude.
- GlobalWeather clients are all over the world and usually consume information using the internet. This is especially useful for transport companies: airlines, bus services, and logistics companies that have a truck fleet to distribute merchandise over a region. Additionally, any companies that coordinate outdoor events.
- Clients need information in real-time.

Case-study: Solving a Business Problem With Service- Oriented Architecture

What's the Business Problem?

GlobalWeather sells information to clients **in real-time**, but it cannot let all clients access their internal systems because of security constraints.

How can we solve this problem?

Let's look at the facts:

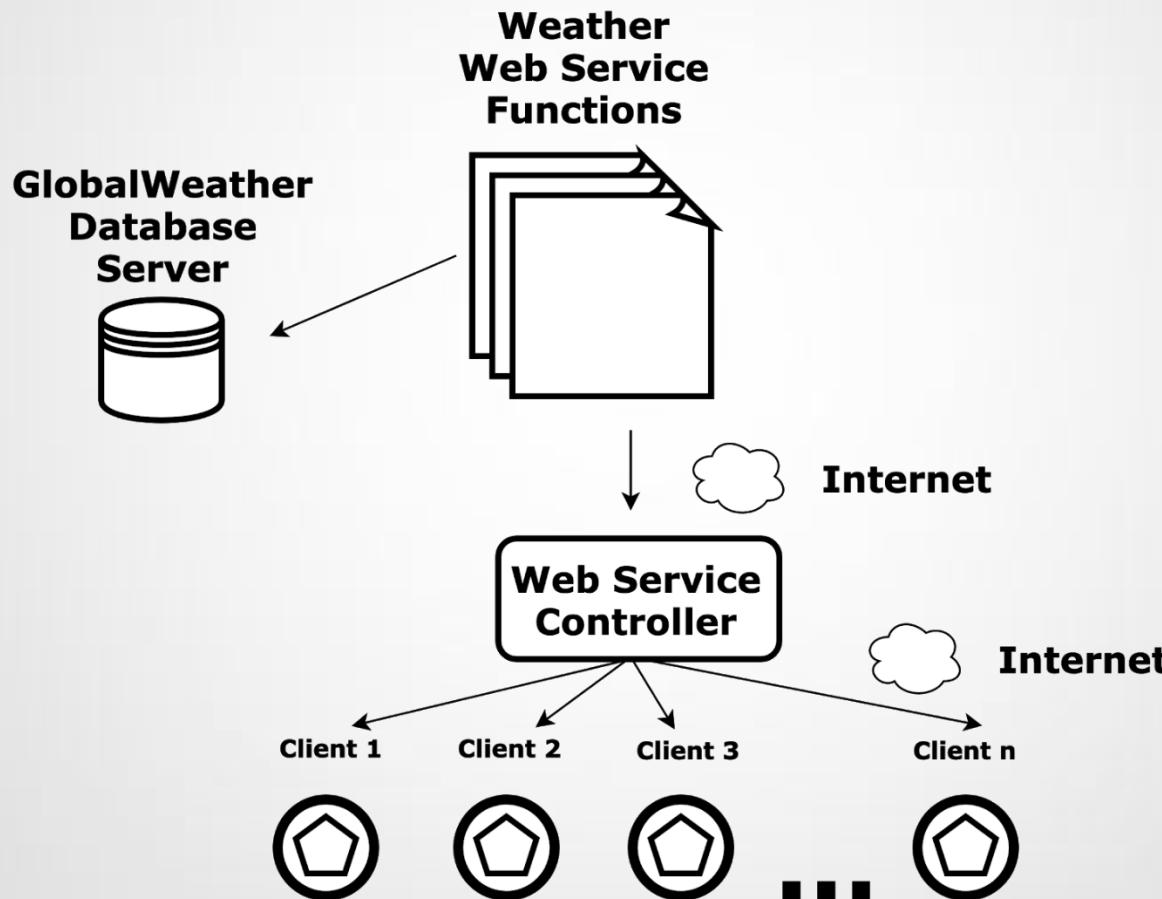
GlobalWeather has **thousands of clients**. To buy its services, each client needs to connect to GlobalWeather systems and get information manually. It is very **complex and time- consuming**, especially since the information requested is usually a little piece of information that can be transmitted via the internet.

Additionally, many clients are **websites that need to connect** to the information feed and quickly display weather information in real-time. For example, an airline that is selling a ticket from city A to city B wants to display the weather in city B once the client buys that ticket.

This situation is the right situation for a **service-oriented architecture**: GlobalWeather can build a **library of functions** and implement it as web services that their clients consume once they pay. This ensures a speedy, real-time service without allowing clients to access internal servers.

What's the Solution?

Service-Oriented Architecture GlobalWeather



Case-study: Solving a Business Problem With Service- Oriented Architecture

Component of the system:

- **Weather web service functions:** This is a library of web services built to serve external websites with weather information such as rain, wind direction and speed, atmospheric pressure, etc.
- **Web service controller:** This module communicates the web service repository with the service requesters. When an external service requester calls a certain function (for instance, flight information, weather information, package status) from the web service repository, the web service controller interprets the call and looks for the function in the web server repository. Then it executes the function and returns a value to the requester.
- **Clients:** These are external applications that request services from the web service repository through the internet. These requests are coded into the requester's application according to a certain syntax published by the service provider.
- **GobalWeather database server:** This server contains the tables, indexes, and data managed by the system.

Case-study: Home Work

You work at a major airport administration department in your country. You have been asked to develop a software system to give the status of a certain flight that is in the air to some possible requesters: airlines, travel sites, hotel sites, etc.

There is a central system at the airport control area that works in real-time: it receives information for the radars and aircrafts as the flights progress.

Now, it's your job to create an architecture for them!

Here are some **key questions** you can ask yourself to get started:

- I. Many flights are in the air at a certain moment. How can you get their flight status from the central system?
- II. How are you going to pass this information to the external requesters?
- III. Why does the airline need this system? Who is going to consume this information?

Can you produce an architecture diagram for this business setting?



CSE- 321

Software Engineering

Development & Coding

Lecture Outlines

- ✧ **Coding**
- ✧ **Coding Standards and Guidelines**
- ✧ **Coding Documentation**
- ✧ **Software Reuse**
- ✧ **Application Frameworks**
- ✧ **Application system reuse**

Coding

Coding

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

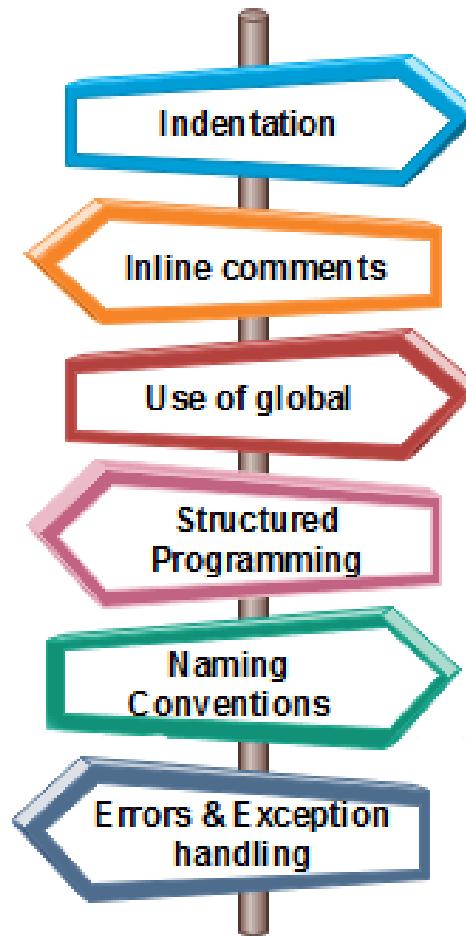
Goals of Coding

- To translate the design of system into a computer language format
- To reduce the cost of later phases
- Making the program more readable

Coding Standards

Coding standards refers to how the developer writes code.

Coding Standards



Coding Standards

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

Indentation should be used to:

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

Inline comments: Inline comments analyse the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.

Rules for limiting the use of global: These rules file what types of data can be declared global and what cannot.

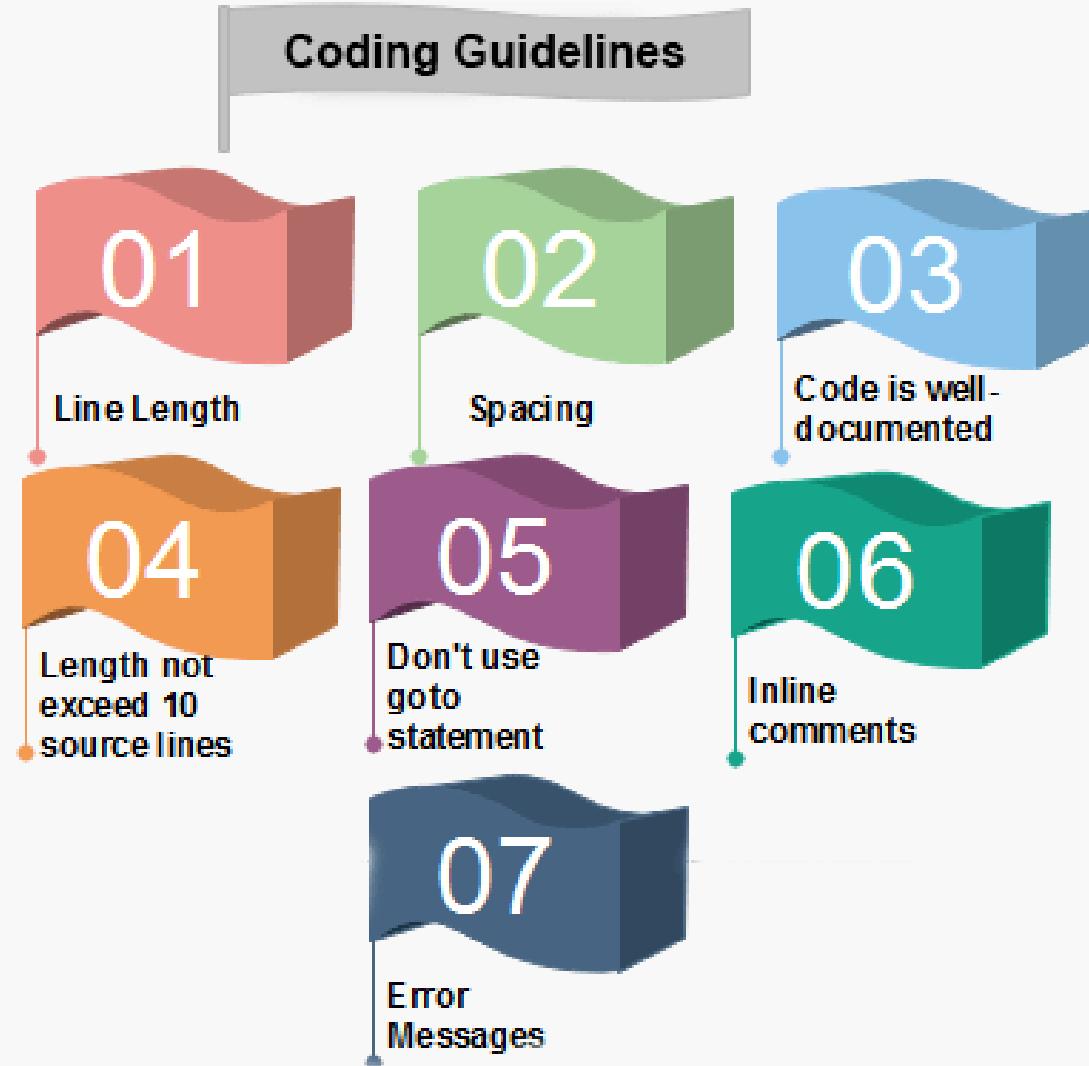
Coding Standards

Structured Programming: Structured (or Modular) Programming methods shall be used.

Naming conventions for global variables, local variables, and constant identifiers: A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

Error return conventions and exception handling system: Different functions in a program report the way error conditions are handled should be standard within an organization.

Coding Guidelines



Coding Guidelines

- 1. Line Length:** It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.
- 2. Spacing:** The appropriate use of spaces within a line of code can improve readability.

Example:

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

Coding Guidelines

- 3. The code should be well-documented:** As a rule of thumb, there must be **at least one comment line on the average for every three-source line.**
- 4. The length of any function should not exceed 10 source lines:** A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.
- 5. Do not use goto statements:** Use of goto statements makes a program unstructured and very tough to understand.
- 6. Inline Comments:** Inline comments promote readability.
- 7. Error Messages:** Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

Coding Guidelines

Naming: In a program, you are required to name the module, processes, and variable, and so on. Care should be taken that the naming style should not be cryptic and non-representative.

For Example: $a = 3.14 * r * r$

```
area_of_circle = 3.14 * radius * radius;
```

Control Constructs: It is desirable that as much as a possible single entry and single exit constructs used.

Information hiding: The information secure in the data structures should be hidden from the rest of the system where possible. Information hiding can decrease the coupling between modules and make the system more maintainable.

Coding Guidelines

Nesting: Deep nesting of loops and conditions greatly harm the static and dynamic behaviour of a program. It also becomes difficult to understand the program logic, so it is desirable to avoid deep nesting.

User-defined types: Make heavy use of user-defined data types like enum, class, structure, and union. These data types make your program code easy to write and easy to understand.

Module size: The **module size should be uniform**. The size of the module should not be too big or too small. If the module size is too large, it is not generally functionally cohesive. If the module size is too small, it leads to unnecessary overheads.

Coding Documentation

Code documentation is a manual-cum-guide that helps in understanding and correctly utilizing the software code. The coding standards and naming conventions written in a commonly spoken language in code documentation provide enhanced clarity for the designer.

Moreover, they act as a guide for the software maintenance team (this team focuses on maintaining software by improving and enhancing the software after it has been delivered to the end user) while the software maintenance process is carried out. In this way, code documentation facilitates code reusability.

Coding Documentation

These are some guidelines for creating the documents –

- Documentation should be from the point of view of the reader
- Document should be unambiguous
- There should be no repetition
- Industry standards should be used
- Documents should always be updated
- Any outdated document should be phased out after due recording of the phase out

Advantages of Documentation

These are some of the advantages of providing program documentation –

- Keeps track of all parts of a software or program
- Maintenance is easier
- Programmers other than the developer can understand all aspects of software
- Improves overall quality of the software
- Assists in user training
- Ensures knowledge de-centralization, cutting costs and effort if people leave the system abruptly

Code Documentation Tools

Code Documentation Tools

While writing software code documentation, it is important to consider the code documentation tools required for writing the software code. The software documentation tools conform to standards by generating the required elements automatically with configurable format and style. These tools combine the selected **comment sections** with the software code to generate a usable documentation with the essential level of details in it.

Code Documentation Tools

Documentation Tools	Language Supported
Cocoon	C++
CcDoc	C++
CxRef	C
DOC++	C, C++, <u>Java</u>
JavaDoc	Java
Perceps	C++
RoboDoc	Assembler, C, Perl, LISP, Fortran, Shell scripts, COBOL
DocJet	Java, C, C++, Visual Basic
ObjectManual	C++
Together	Java, C++
Doc-o-matic	C++, C#, ASP.NET, VB.NET, Java, JavaScript, JSP

Software reuse

- In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.
- Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need a design process that is based on systematic software reuse.
- There has been a major switch to reuse-based development over the past 10 years.
- **Software reuse** is the process of implementing or updating software systems using existing software assets.

Reuse-based software engineering

- System reuse
 - Complete systems, which may include several application programs may be reused.
- Application reuse
 - An application may be reused either by incorporating it without change into other or by developing application families.
- Component reuse
 - Components of an application from sub-systems to single objects may be reused.
- Object and function reuse
 - Small-scale software components that implement a single well-defined object or function may be reused.

Benefits of software reuse

- Accelerated development
- Effective use of specialists
- Increased dependability
- Lower development costs
- Reduced process risk
- Creating, maintaining, and using a component library
- Finding, understanding, and adapting reusable components

Problems with software reuse

Creating, maintaining, and using a component library: Populating a reusable component library and ensuring the software developers can use this library can be expensive. Development processes have to be adapted to ensure that the library is used

Finding, understanding, and adapting reusable components: Software components have to be discovered in a library, understood and, sometimes, adapted to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they include a component search as part of their normal development process

Increased maintenance costs: If the source code of a reused software system or component is not available then maintenance costs may be higher because the reused elements of the system may become increasingly incompatible with system changes.

Problems with software reuse

Lack of tool support: Some software tools do not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account. This is particularly true for tools that support embedded systems engineering, less so for object-oriented development tools.

Not-invented-here syndrome: Some software engineers prefer to **rewrite components** because they believe they can improve on them. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.

Approaches that support software reuse

Approach
Application frameworks
Application system integration
Design patterns
Architectural patterns
Aspect-oriented software development
Configurable application systems
Legacy system wrapping
Model-driven engineering
Program generators
Program libraries
Service-oriented systems

Reuse planning factors

Key factors for reuse planning:

- ❖ The development schedule for the software.
- ❖ The expected software lifetime.
- ❖ The background, skills and experience of the development team.
- ❖ The criticality of the software and its non-functional requirements.
- ❖ The application domain.
- ❖ The execution platform for the software.

Application system reuse



Application system reuse

- ✧ An **application system product** is a software system that can be adapted for different customers **without changing the source code** of the system.
 - ✧ Application systems have **generic features** and so can be used/reused in different environments.
 - ✧ Application system products are adapted by using built in configuration mechanisms that allow the functionality of the system to be tailored to specific customer needs.
- For example, in a hospital patient record system, separate input forms and output reports might be defined for different types of patient.

Application system reuse

Benefits of application system reuse:

- ✧ As with other types of reuse, **more rapid deployment** of a reliable system may be possible.
- ✧ It is possible to see what functionality is provided by the applications and so it is easier to judge whether or not they are likely to be suitable.
- ✧ Some development **risks are avoided** by using existing software. However, this approach has its own risks, as I discuss below.
- ✧ Businesses can **focus on their core activity** without having to devote a lot of resources to IT systems development.
- ✧ As operating platforms evolve, technology updates may be simplified as these are the responsibility of the **commercial-off-the-shelf (COTS)** product vendor rather than the customer.

Application system reuse

Problems of application system reuse:

- ✧ Requirements usually have to be adapted to reflect the functionality and mode of operation of the COTS product.
- ✧ The COTS product may be based on assumptions that are practically impossible to change.
- ✧ **Choosing the right COTS system** for an enterprise can be a difficult process, especially as many COTS products are not well documented.
- ✧ There may be a **lack of local expertise** to support systems development.
- ✧ The COTS product vendor controls system support and evolution.

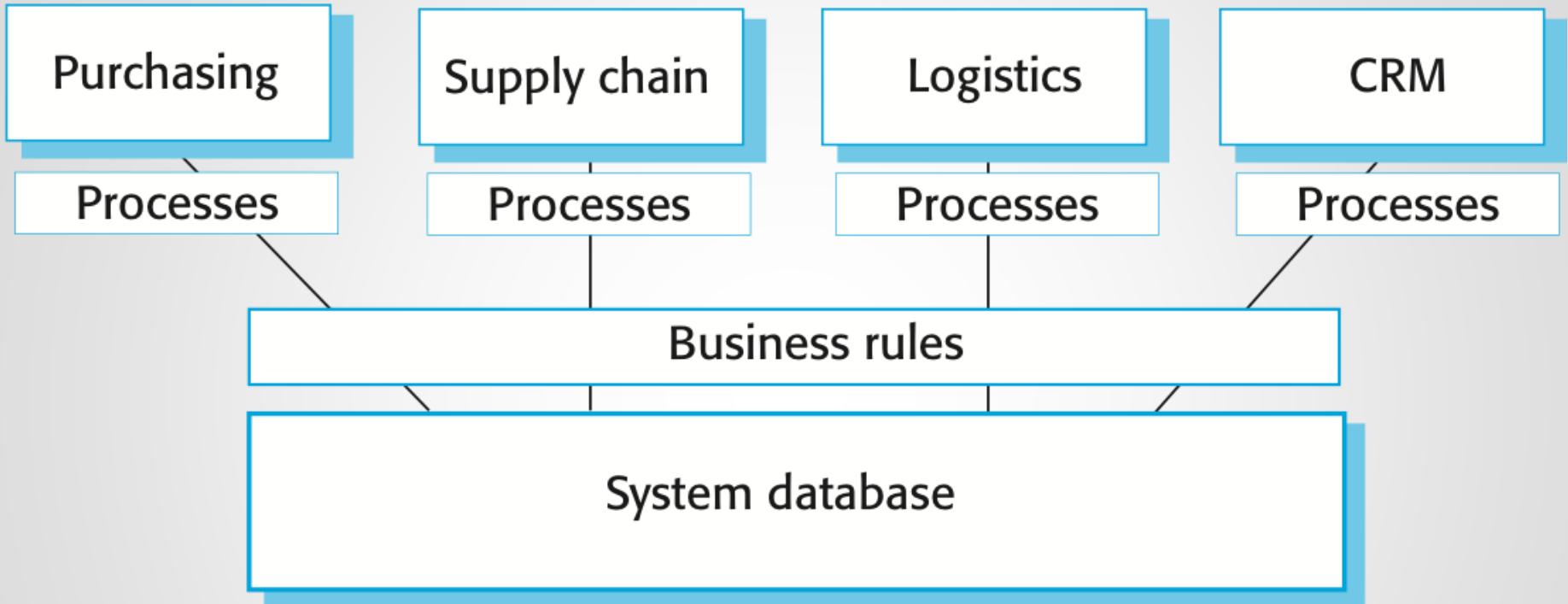
Configurable application systems

- ✧ **Configurable application systems** are generic application systems that may be designed to **support a particular** business type, business activity or, sometimes, a complete business enterprise.
- ✧ For example, an application system may be produced for dentists that handles appointments, dental records, patient recall, etc.
- ✧ Domain-specific systems, such as systems to support a business function (e.g. document management) provide functionality that is likely to be required by a range of potential users.

Enterprise Resource Planning (ERP)

- ✧ An Enterprise Resource Planning (ERP) system is a generic system that supports **common business processes** such as ordering and invoicing, manufacturing, etc.
- ✧ These are very widely used in large companies - they represent probably the most common form of software reuse.
- ✧ The generic core is adapted by including modules and by incorporating knowledge of business processes and rules.
- ✧ A number of modules to support different business functions. A defined set of business processes, associated with each module, which relate to activities in that module.
- ✧ A common database that maintains information about all related business functions. A set of business rules that apply to all data in the database.

Enterprise Resource Planning (ERP)



Key elements of an ERP system architecture:

- ✧ A number of modules to **support different business functions**.
- ✧ A defined set of business processes, associated with each module, which relate to activities in that module.
- ✧ A **common database** that maintains information about all related business functions.
- ✧ A **set of business rules** that apply to all data in the database.

ERP system configuration

An ERP system configuration usually involves:

- ✧ Selecting the required functionality from the system.
- ✧ Establishing a data model that defines how the organization's data will be structured in the system database.
- ✧ Defining business rules that apply to that data.
- ✧ Defining the expected interactions with external systems.
- ✧ Designing the input forms and the output reports generated by the system.
- ✧ Designing new business processes that conform to the underlying process model supported by the system.
- ✧ Setting parameters that define how the system is deployed on its underlying platform.

Question

The reuse of software raises a number of copyright and intellectual property issues. If a customer pays a software contractor to develop a system.

- Who has the right to reuse the developed code?
- Does the software contractor have the right to use that code as a basis for a **generic component**?
- What payment mechanisms might be used to reimburse providers of reusable components?
- Discuss these issues and other ethical issues associated with the reuse of software.

Solution:

<https://paigepecksite.wordpress.com/2016/09/29/hw15-chapter-15/>

Question

Because the **customer** paid the software contractor specifically to develop this specific system, the customer should have the **rights to reuse** the developed code.

When a **software developer** is working for a company designing code, it is my understanding that the code they write is owned by the company, unless it is an open source platform. With that being said, With that being said, a software developer has learned coding techniques and how to do algorithms in a proficient manner, so **the generic code itself could be reused** as long as they aren't copy and pasting it from the actual developed code.

They would have to rewrite it as a whole new system and be wary not to be designing something that would be considered in direct competition of the software they developed for the previous company. This leaves a very gray line that could easily be misused, but I don't see how a software developer could not reuse some components especially if they are in the same field but with a different company.

Question

While there are things such as drawings, poems, beats to a song that can be copyrighted because they can be unique, software is a different entity entirely.

As for a payment mechanism for the reimbursement of reusable components, this is a difficult one. Again, it is not like a software developer should just stop using an algorithm because they created it for a different company when it can help them accomplish what they need, but if it is a component that can be “sold”, then perhaps it could be sold like assets are sold for 3D modeling online. It is a very gray area that is difficult to pinpoint what should/could be done, but it is of my opinion that a software developer should be allowed to better themselves without worrying about copyright and intellectual property issues.

Which means in the grand scheme of things, it should not be the customer who is allowed to sell the reusable components as they paid the contractor to write it, not to create this system for only them alone which should be a higher cost if such a thing were to happen.



CSE- 321

Software Engineering

Software Cost Estimation

Software Cost Estimation



Software Cost Estimation

Estimation is the process of **finding an estimate, or approximation**, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

Estimation determines **how much money, effort, resources, and time** it will take to build a specific system or product.

Estimation is based on –

1. Past Data/Past Experience
2. Available Documents/Knowledge
3. Assumptions
4. Identified Risks

Fundamental estimation questions

1. How much **effort** is required to complete an activity?
2. How much calendar **time** is needed to complete an activity?
3. What is the **total cost** of an activity?
4. Project estimation and scheduling and interleaved **management activities**

Software Cost Estimation

Software Cost Estimation

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated, **but how is this done?**

Several estimation procedures have been developed and are having the **following attributes in common.**

- Project scope must be established in **advanced**.
- Software metrics are used as a **support from** which evaluation is made.
- The project is **broken into small PCs** which are estimated individually.
To achieve true cost & schedule estimate, several option arise.
- **Delay estimation**
- Used symbol **decomposition techniques** to generate project cost and schedule estimates.
- Acquire one or more **automated estimation tools.**

Software cost components

Software cost components

- Hardware and software costs
- Travel and training costs
- Effort costs (the dominant factor in most projects)
 - salaries of engineers involved in the project
 - Social and insurance costs

This following costs are all part of the **total effort cost**:

- Costs of providing, heating and lighting office space
- Costs of support staff such as accountants, administrators, system managers, cleaners and technicians
- Costs of networking and communications
- Costs of central facilities such as a library or recreational facilities
- Costs of Social Security and employee benefits such as pensions and health insurance.

Factors that affecting software pricing

Factor	Description
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business.

Estimation techniques

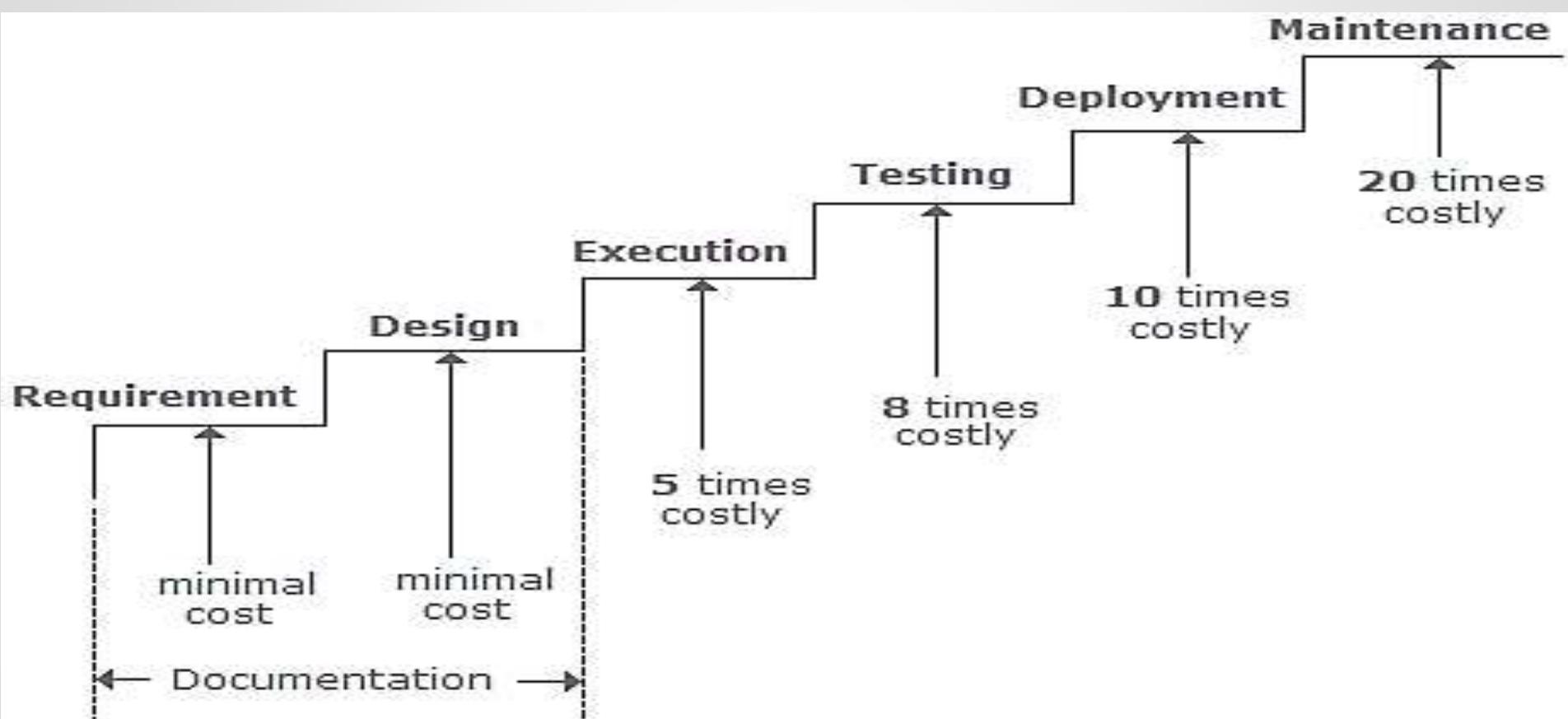
- There is no simple way to make an accurate estimate of the effort required to develop a software system
 - Initial estimates are based on inadequate information in a user requirements definition
 - The software may run on unfamiliar computers or use new technology
 - The people in the project may be unknown
- Project cost estimates may be self-fulfilling
 - The estimate defines the budget and the product is adjusted to meet the budget

Estimation techniques

- Algorithmic cost modelling
- Expert judgement
- Estimation by analogy
- Parkinson's Law
- Pricing to win

Estimation techniques error effects

A defect, which could have been removed during the initial stage, is removed in a later stage. How does this affect the cost?



COCOMO model



- LOC: Lines of Code
- Function Point
- Cost Estimation
- COCOMO I
 - Basic COCOMO
 - Intermediate COCOMO
 - Detailed COCOMO
- COCOMO II

LINES OF CODE (LOC)

- **Lines of code(LOC) or Source lines of code (SLOC)**is a software metric used to measure the **size of a software program** by counting the number of lines in the text of the program's source code.
- LOC is typically used **to predict the amount of effort that will be required to develop a program**, as well as to estimate programming productivity or effort once the software is produced.

FUNCTION POINT

A **Function Point (FP)** is a unit of measurement **to express the amount of business functionality**, an information system (as a product) provides to a user. FPs **measure software size**. They are widely accepted as an industry standard for **functional sizing**.

Constructive Cost Model(COCOMO)

- The COCOMO model is an **algorithmic software** cost estimation model.
- It was proposed by **Barry Boehm** in 1970 and is based on the study of **63 projects**, which make it one of the best-documented models.
- The model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics.
- The COCOMO estimates the cost for software product development in terms of **effort** (resources required to complete the project work) and **schedule** (time required to complete the project work) based on the size of the software product.
- It estimates the required number of **Man-Months** (MM) for the full development of software products

Constructive Cost Model(COCOMO)

COCOMO model are categorized into three types:

1. Basic COCOMO
2. Intermediate COCOMO
3. Detailed COCOMO

According to COCOMO, there are three modes of software development projects that depend on complexity.

Such as:

1. **Organic Project**
2. **Semidetached Project**
3. **Embedded Project**
- .

Development Mode of S/W Projects

Mode	Project Size	Nature of Project	Innovation	Deadline
Organic	Typically 2-50 KLOC	Small size project, Experienced developers.	Little	Not Tight
Semi Detached	Typically 50-300KLOC	Medium size project and team.	Medium	Medium
Embedded	Typically over 300KLOC	Large project, Real-time systems	Significant	Tight

KLOC is a measure of the size of a computer program. The size is determined by measuring the number of lines of source code a program has.

Constructive Cost Model(COCOMO)

1. Organic Project

It belongs to small & simple software projects which are handled by a small team with good domain knowledge and few rigid requirements.

Example: Small data processing or Inventory management system.

2. Semidetached Project

It is an intermediate (in terms of size and complexity) project, where the team having mixed experience (both experience & inexperience resources) to deals with rigid/nonrigid requirements.

Example: Database design or OS development.

3. Embedded Project

This project having a high level of complexity with a large team size by considering all sets of parameters (software, hardware and operational).

Example: Banking software or Traffic light control software.

Basic COCOMO Model

The Basic COCOMO

It is the one type of static model to estimates software development effort quickly and roughly. It mainly deals with the **number of lines** of code and the level of estimation accuracy is less as **we don't consider the all parameters belongs to the project**. The estimated **effort and scheduled time** for the project are given by the relation:

$$\text{Effort (E)} = a * (\text{KLOC})^b \text{ MM}$$

$$\text{Scheduled Time (D)} = c * (E)^d \text{ Months(M)}$$

Basic COCOMO Model

Where,

E = Total effort required for the project in Man-Months (MM).

D = Total time required for project development in Months (M).

KLOC = the size of the code for the project in Kilo lines of code.

a, b, c, d = The constant parameters for a software project.

PROJECT TYPE	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

Basic cocomo: Example-1

Example: For a given project was estimated with a size of 300 KLOC. Calculate the Effort, Scheduled time for development. Also, calculate the Average resource size and Productivity of the software for Organic project type.

Ans: Given estimated size of project is: 300 KLOC

For Organic

$$\text{Effort (E)} = a * (\text{KLOC})^b = 2.4 * (300)^{1.05} = 957.61 \text{ MM}$$

$$\text{Scheduled Time (D)} = c * (E)^d = 2.5 * (957.61)^{0.38} = 33.95 \text{ Months(M)}$$

$$\text{Avg. Resource Size} = E/D = 957.61 / 33.95 = 28.21 \text{ Mans}$$

$$\text{Productivity of Software} = \text{KLOC/E} = 300 / 957.61 = 0.3132 \text{ KLOC/MM} = 313 \text{ LOC/MM}$$

Basic cocomo: Example-1

Example: For a given project was estimated with a size of 300 KLOC. Calculate the Effort, Scheduled time for development. Also, calculate the Average resource size and Productivity of the software for Organic project type.

Ans: Given estimated size of project is: 300 KLOC

For Semidetached

$$\text{Effort (E)} = a * (\text{KLOC})^b = 3.0 * (300)^{1.12} = 1784.42 \text{ MM}$$

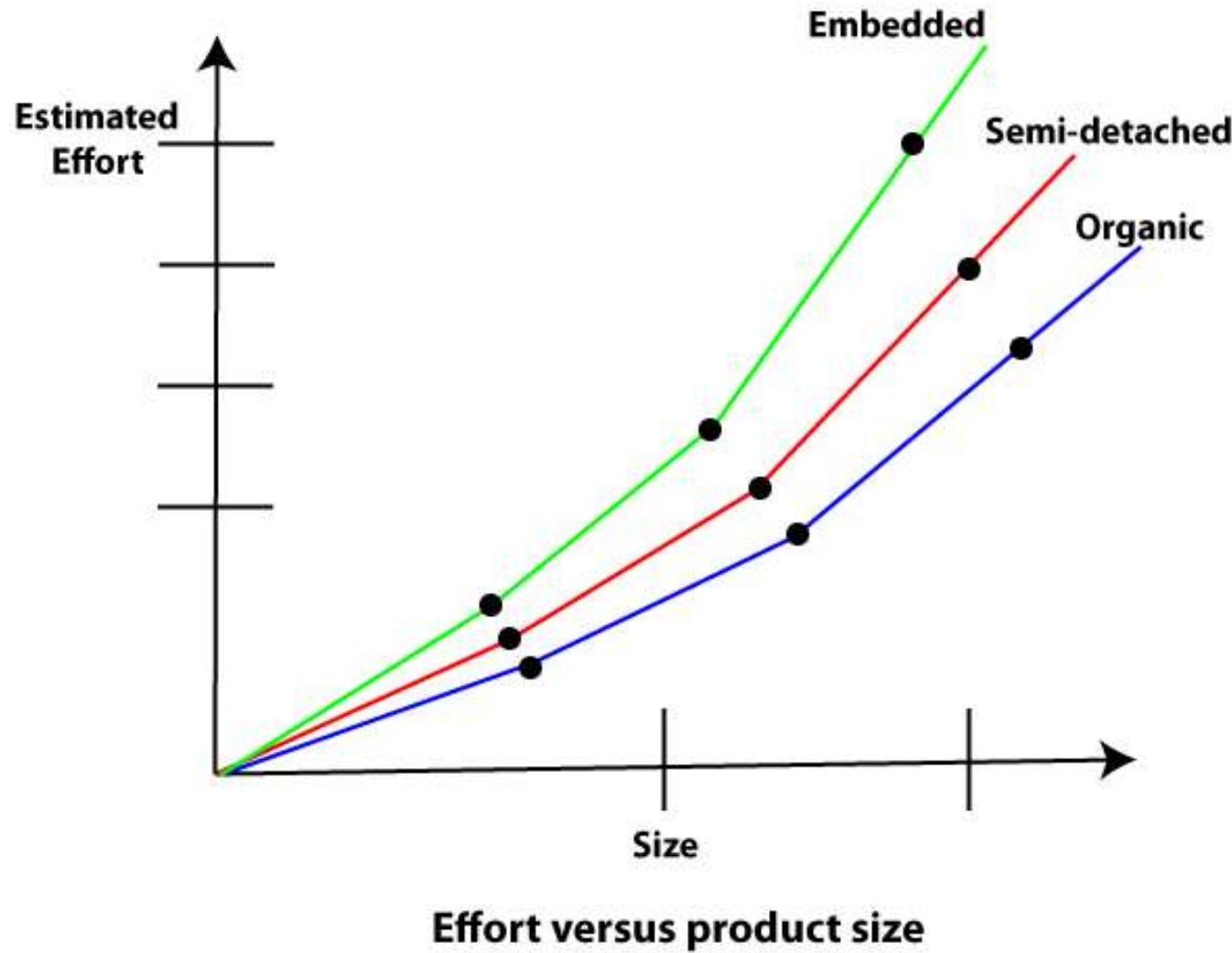
$$\text{Scheduled Time (D)} = c * (E)^d = 2.5 * (1784.42)^{0.35} = 34.35 \text{ Months(M)}$$

For Embedded

$$\text{Effort (E)} = a * (\text{KLOC})^b = 3.6 * (300)^{1.2} = 3379.46 \text{ MM}$$

$$\text{Scheduled Time (D)} = c * (E)^d = 2.5 * (3379.46)^{0.32} = 33.66 \text{ Months(M)}$$

development time versus the product size



Basic cocomo: Example-2

Example: A project size of 200KLOC is to be developed. S/W development team has average experience on similar type of projects. The project schedule is not very tight (that means medium). Calculate the effort and development time of the project.

Ans: 200 KLOC implies semi-detached mode.

Hence,

$$E = 3.0 * (200)^{1.12} = 1133.12 \text{ MM}$$

$$D = 2.5 * (1133.12)^{0.35} = 29.3 \text{ M}$$

Avg. staff size(SS) = E/D = $1133.12/29.3=38.67$ Persons.

Productivity (P) = KLOC/E = $200/1133.12=0.1765$ KLOC/MM.

Intermediate COCOMO

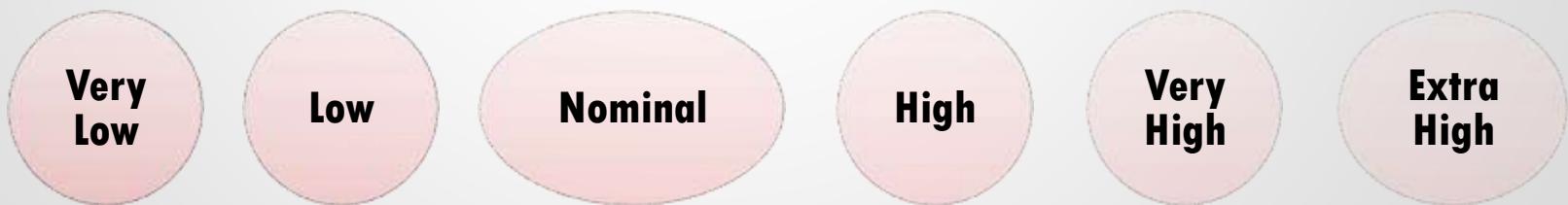
The intermediate model estimates software development effort in terms of size of the program and other related **cost drivers parameters** (product parameter, hardware parameter, resource parameter, and project parameter) of the project.

Classification of Cost Drivers :

- (i) Product attributes
- (ii) Computer attributes
- (iii) Personnel attributes
- (iv) Project attributes

Intermediate COCOMO

- Extension of Basic COCOMO
- **Why Use ?**
 - Basic model lacks accuracy
- Computes software development effort as a function of program size and set of 15 Cost Drivers
- **Cost Driver:** A multiplicative factor that determines the effort required to complete the software project.
- **Why Cost Drivers?**
 - Adjust the nominal cost of a project to the actual project Environment.
- For each Characteristics, Estimator decides the scale factor



Cost Drivers

Product Attributes

- Required Software Reliability (RELY)
- Database Size (DATA)
- Product Complexity (CPLX)

Computer Attributes

- Execution Time Constraint (TIME)
- Main Storage constraint (STOR)
- Virtual Machine volatility (VIRT)
- Computer turnaround time (TURN)

Personnel Attributes

- Analyst Capability (ACAP)
- Application Experience (AEXP)
- Programmer Capability (PCAP)
- Virtual Machine Experience (VEXP)
- Programming language Experience (LEXP)

Project Attributes

- Modern programming practices (MODP)
- Use of Software tools (TOOL)
- Required development schedule (SCED)

The Calculation

- Multiply all 15 Cost Drivers to get **Effort Adjustment Factor(EAF)**
- $E(\text{Effort}) = a(\text{KLOC})^b * \text{EAF}(\text{in Person-Month})$
- $D(\text{Development Time}) = c(E)^d (\text{in month})$
- $SS (\text{Avg Staff Size}) = E/D (\text{in persons})$
- $P (\text{Productivity}) = \text{KLOC}/E (\text{in KLOC/Person-month})$

Project	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Intermediate COCOMO : Example

A new project with estimated **400 KLOC embedded system** has to be developed. Project manager hires developers of very **low quality** but a **lot of experience** in programming language. Calculate the Effort, Development time, Staff size & Productivity.

Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
AEXP	1.29	1.13	1.00	0.91	0.82	--
LEXP	1.14	1.07	1.00	0.95	--	--

$$EAF = 1.29 * 0.95 = 1.225$$

400 LOC implies Embedded System

$$\text{Effort} = 2.8 * (400)^{1.20} * 1.225 = 3712 * 1.22 = 4528 \text{ person-months}$$

$$\text{Development Time} = 2.5 * (4528)^{0.32} = 2.5 * 14.78 = 36.9 \text{ months}$$

$$\text{Avg. Staff Size} = E/D = 4528/36.9 = 122 \text{ persons}$$

$$\text{Productivity} = \text{KLOC/Effort} = 400/4528 = 0.0884 \text{ KLOC/person-month}$$

Project	a	b	c	d
Embedded	2.8	1.20	2.5	0.32

Detailed COCOMO =

Intermediate COCOMO + assessment of Cost Drivers impact on each phase.

- Phases
 - 1) Plans and requirements
 - 2) System Design
 - 3) Detailed Design
 - 4) Module code and test
 - 5) Integrate and test

Cost of each subsystem is estimated separately. This reduces the margin of error.

Advantages and Disadvantages of COCOMO Model

Advantages

- Easy to estimate the total cost of the project.
- Easy to implement with various factors.
- Provide ideas about historical projects.

Disadvantages

- It ignores requirements, customer skills, and hardware issues.
- It limits the accuracy of the software costs.
- It mostly depends on time factors.

WHY COCOMO-II ?

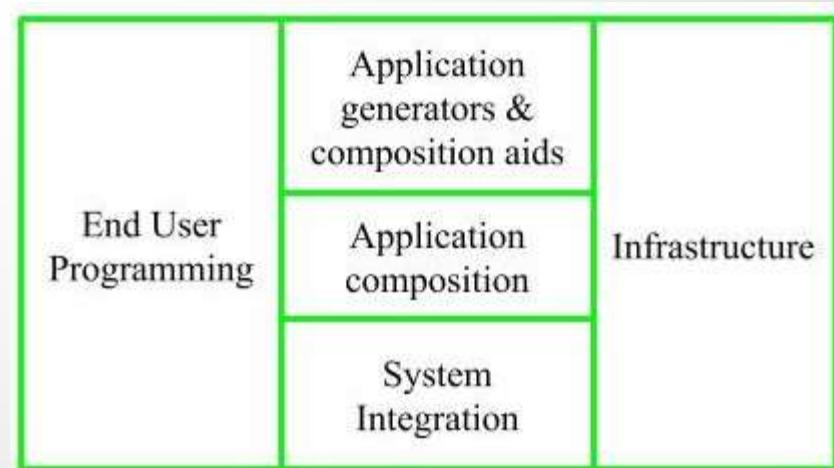
- The changes in s/w development techniques included a move away from mainframe overnight batch processing to desktop-based real-time turnaround.
- These changes and others began to make applying the original COCOMO model problematic.
- The model is tuned to the life cycle practices of the 21st century.

COCOMO-II is the revised version of the original COCOMO (Constructive Cost Model) and is developed at **University of Southern California**. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.

It consists of three sub-models:

Stages Of Cocomo-II

1. Application Composition
2. Earlier Design
3. Post Architecture



Final Word

“The models are just there to help, not to make the management decisions for you.”

--Barry Boehm



CSE- 321

Software Engineering

**Software Verification &
Validation , Testing, Quality**

Lecture Outlines

- ✧ **Verification and Validation**
- ✧ **Cleanroom software development**
- ✧ **Software testing**
- ✧ **Software testing types**
- ✧ **Black-box Testing ,White-box Testing, Grey-Box Testing**

Verification and Validation

Verification and Validation

Verification:

"Are we building the product right".

The software should conform to its specification.

Validation:

"Are we building the right product".

The software should do what the user really requires.

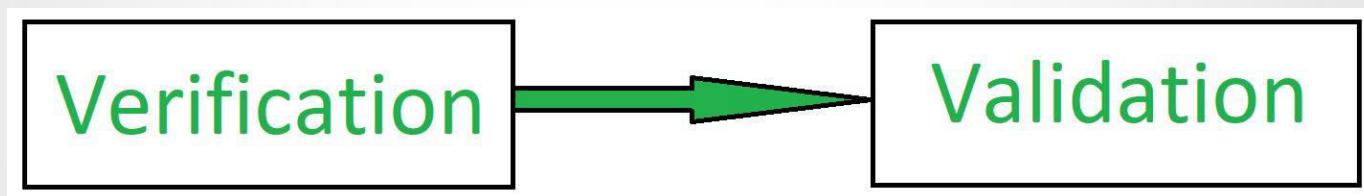


Verification and Validation (V & V)

Verification:

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

Activities involved in verification: Inspections Reviews, Walkthroughs, Desk-checking



Validation:

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product , it checks what we are developing **is the right product**. it is validation of actual and expected product.

Includes program reviews, system testing, customer acceptance testing.

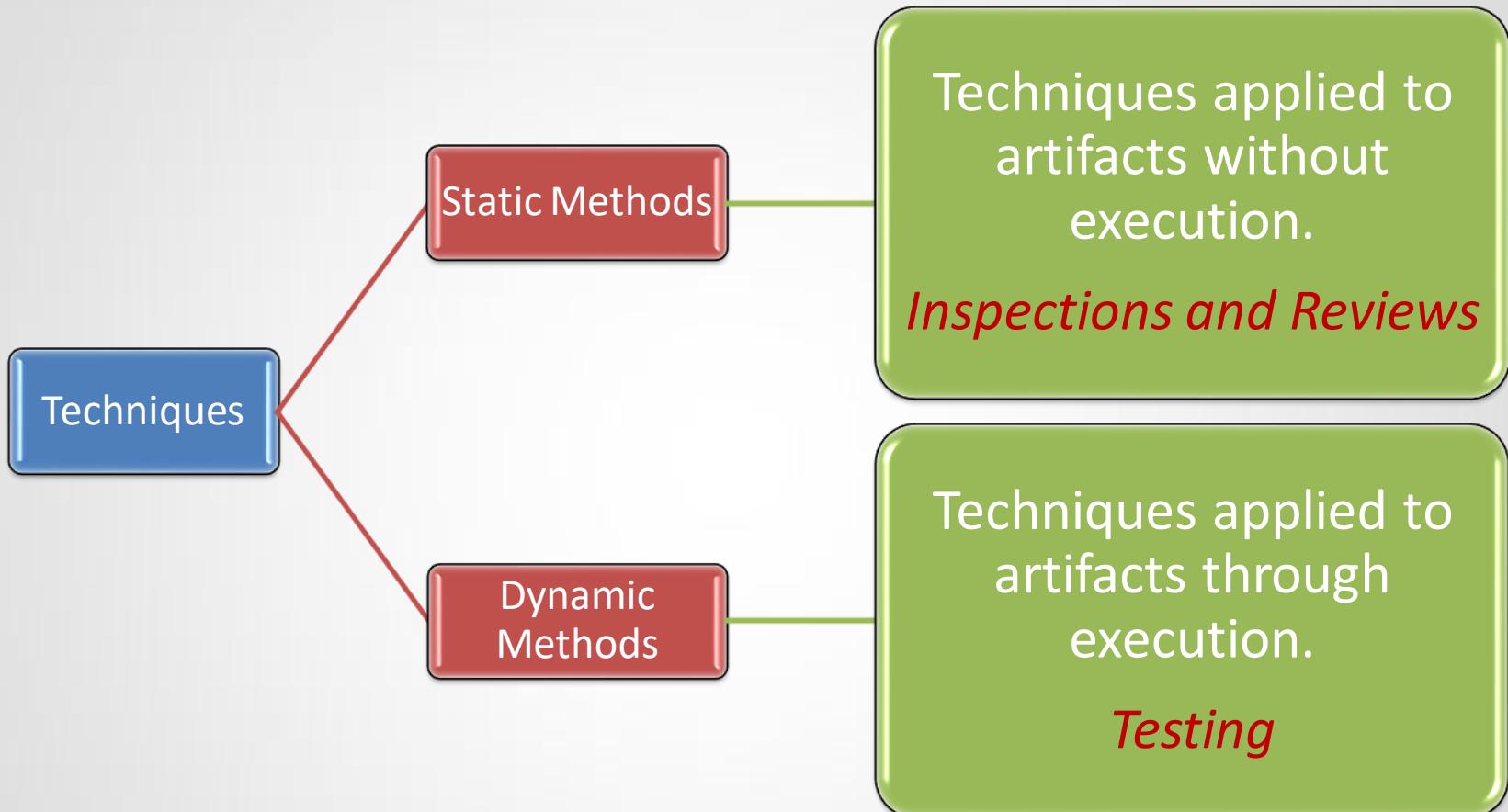
The V & V process

- Is a whole life-cycle process - V & V must be applied at each stage in the software process.
- Has two principal objectives
 - The **discovery of defects** in a system;
 - The **assessment** of whether or not the system is **useful** and **useable** in an operational situation.

V & V goals

- Verification and validation should establish confidence that the software is **fit for purpose**.
- This does **NOT** mean completely free of defects.
- Rather, it must be good enough for its intended use and the **type of use** will determine the **degree of confidence** that is needed.

V and V Techniques



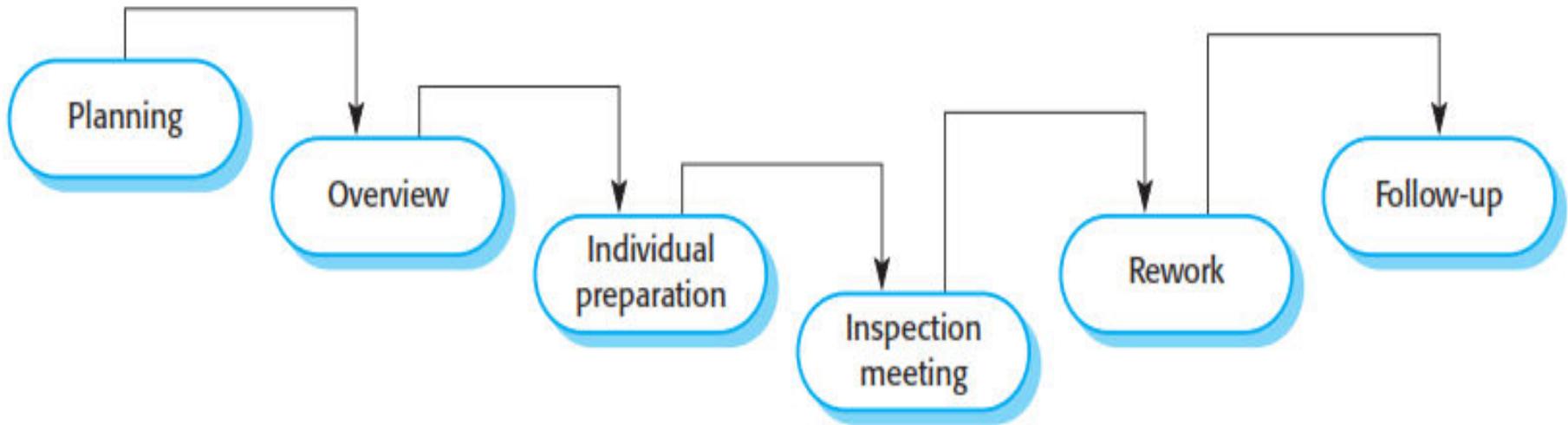
Static and dynamic verification

- Software inspections is a static V & V process in which a software system is reviewed to find errors, omissions and anomalies. Generally, inspections focus on source code, but any readable representation of the software such as its requirements or a design model can be inspected.
 - May be supplement by tool-based document and code analysis
 - Inspections do not require execution of a system so may be used before implementation.
 - They may be applied to any representation of the system such as the requirements or design.
 - They have been shown to be an effective technique for discovering program errors.

Static and dynamic verification

- Software inspections
 - Inspections are an old idea. There have been several studies and experiments that have demonstrated that inspections are more effective for defect discovery than program testing.
 - Fagan (Fagan, 1986) reported that more than 60% of the errors in a program can be detected using informal program inspections.
 - Mills et al. (Mills, et al., 1987) suggest that a more formal approach to inspection based on correctness arguments can detect more than 90% of the errors in a program.

The inspection process



- System overview presented to inspection team.
- Code and associated documents are distributed to inspection team in advance for individual preparation.
- Inspection meeting takes place and discovered errors are noted.
- Modifications are made to repair discovered errors (by owner).
- Re-inspection may or may not be required.

Inspection Checks

Possible checks that might be made during the inspection process

Fault class	Inspection check
Data faults	<p>Are all program variables initialised before their values are used?</p> <p>Have all constants been named?</p> <p>Should the upper bound of arrays be equal to the size of the array or Size -1?</p> <p>If character strings are used, is a delimiter explicitly assigned?</p> <p>Is there any possibility of buffer overflow?</p>
Control faults	<p>For each conditional statement, is the condition correct?</p> <p>Is each loop certain to terminate?</p> <p>Are compound statements correctly bracketed?</p> <p>In case statements, are all possible cases accounted for?</p> <p>If a break is required after each case in case statements, has it been included?</p>
Input/output faults	<p>Are all input variables used?</p> <p>Are all output variables assigned a value before they are output?</p> <p>Can unexpected inputs cause corruption?</p>

Inspection Checks

Possible checks that might be made during the inspection process

Fault class	Inspection check
Interface faults	<p>Do all function and method calls have the correct number of parameters?</p> <p>Do formal and actual parameter types match?</p> <p>Are the parameters in the right order?</p> <p>If components access shared memory, do they have the same model of the shared memory structure?</p>
Storage management faults	<p>If a linked structure is modified, have all links been correctly reassigned?</p> <p>If dynamic storage is used, has space been allocated correctly?</p> <p>Is space explicitly de-allocated after it is no longer required?</p>
Exception management faults	Have all possible error conditions been taken into account?

Static and dynamic verification

- **Software testing** Concerned with exercising and observing product behaviour (dynamic verification)
 - Inspections and testing are complementary and not opposing verification techniques. Both should be used during the V & V process.
 - Inspections can check conformance with a specification but not conformance with the customer's real requirements. Inspections cannot check **non-functional characteristics** such as performance, usability, etc.
 - Software testing is a process of identifying the **correctness of software by considering its all attributes** (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.

Program testing

- Can reveal the presence of errors **NOT** their absence.
- The only validation technique for **non-functional requirements** as the software has to be executed to see **how it behaves**.
- Should be **used in conjunction** with static verification to provide full V&V coverage
- Typically, a commercial software system has to go through three stages of testing:
 - **Development testing:** the system is tested during development to discover bugs and defects.
 - **Release testing:** a separate testing team test a complete version of the system before it is released to users.
 - **User testing:** users or potential users of a system test the system in their own environment.

Verification vs Validation

Verification	Validation
Are we implementing the system right?	Are we implementing the right system?
Evaluating products of a development phase	Evaluating products at the closing of the development process
The objective is making sure the product is as per the requirements and design specifications	The objective is making sure that the product meets user's requirements
Activities included: reviews, meetings, and inspections	Activities included: black box testing, white box testing, and grey box testing
Verifies that outputs are according to inputs or not	Validates that the users accept the software or not
Items evaluated: plans, requirement specifications, design specifications, code, and test cases	Items evaluated: actual product or software under test
Manual checking of the documents and files	Checking the developed products using the documents and files

Verification vs Validation

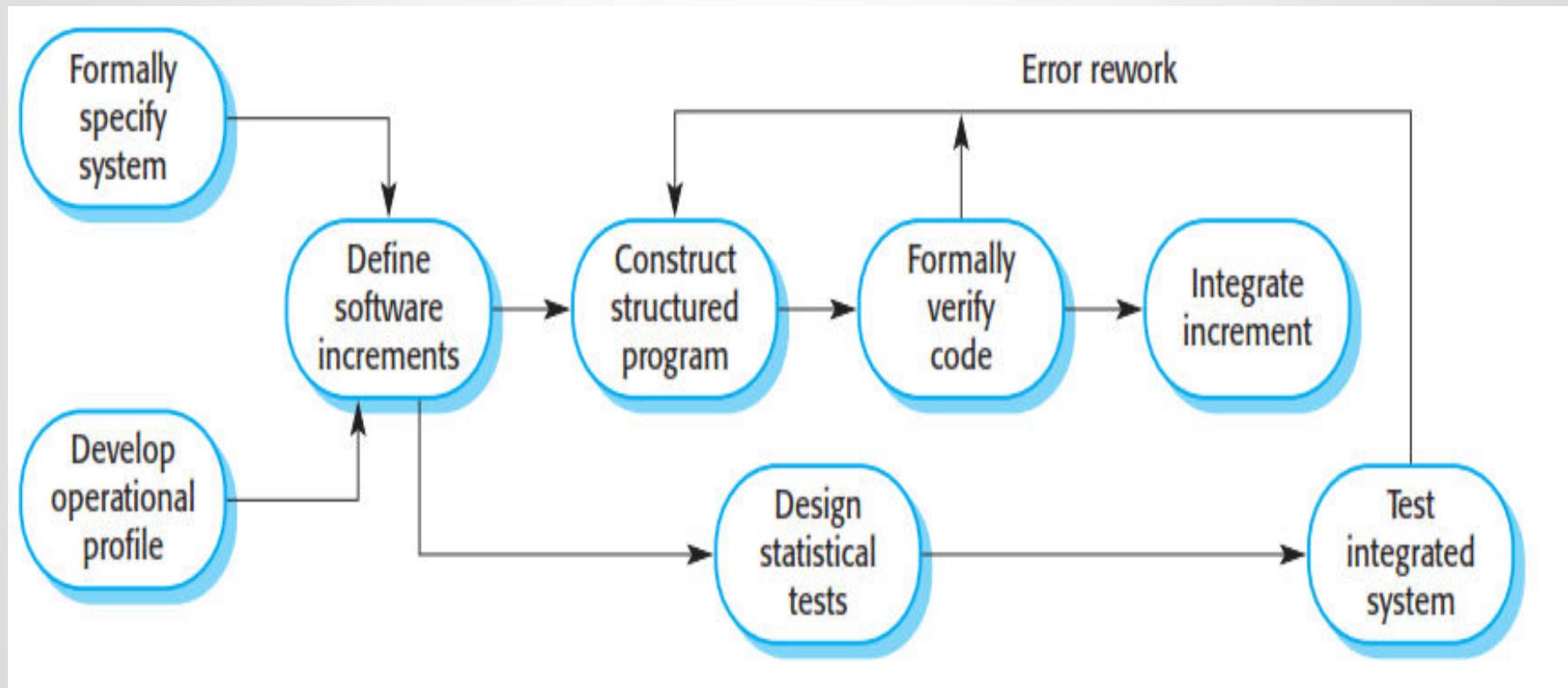
VERIFICATION	VALIDATION
Verification is the static testing.	Validation is the dynamic testing.
It does not include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.

Cleanroom software development

- The name is derived from the “Cleanroom” process in semiconductor fabrication.
- **The philosophy is defect avoidance rather than defect removal.**
- Cleanroom software development (Mills, et al., 1987; Cobb and Mills, 1990; Linger, 1994; Prowell, et al., 1999) is a software development philosophy that uses formal methods to support rigorous software inspection.
- A software development process based on:
 - Incremental development (if appropriate)
 - Formal specification
 - Static verification using correctness arguments
 - Statistical testing to certify program reliability

The Cleanroom process

The objective of this approach to software development is zero-defect software.



The Cleanroom process

The Cleanroom approach to software development is based on five key strategies:

1. **Formal specification** The software to be developed is formally specified. A state transition model that shows system responses to stimuli is used to express the specification.
2. **Incremental development** The software is partitioned into increments that are developed and validated separately using the Cleanroom process. These increments are specified, with customer input, at an early stage in the process.
3. **Structured programming** Only a limited number of control and data abstraction constructs are used. The program development process is a process of stepwise refinement of the specification. A limited number of constructs are used and the aim is to systematically transform the specification to create the program code.

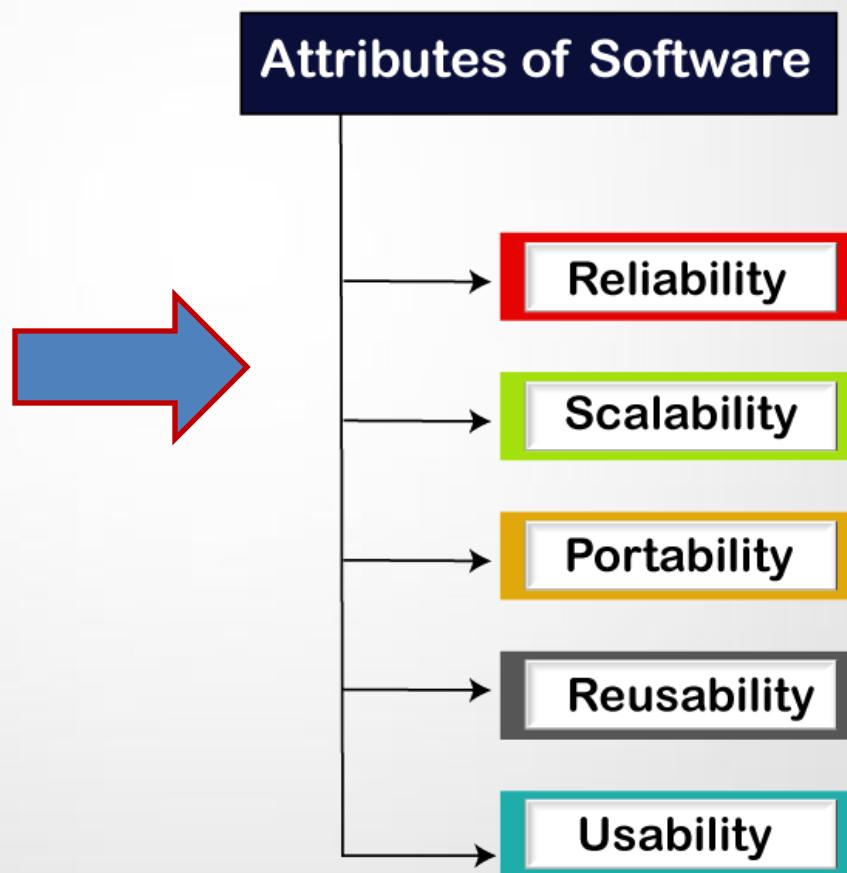
The Cleanroom process

4. **Static verification** The developed software is statically verified using rigorous software inspections. There is no unit or module testing process for code components.
5. **Statistical testing of the system** The integrated software increment is tested statistically, to determine its reliability. These statistical tests are based on an operational profile, which is developed in parallel With the system specification.

Software testing

Software testing

Software testing is a process of identifying the **correctness** of software by considering its all attributes (**Reliability, Scalability, Portability, Re-usability, Usability**) and evaluating the execution of software components to find the software bugs or errors or defects.



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.

Software Testing Definition according to **ANSI/IEEE 1059** standard – A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

Testing can only show the **presence of errors** in a program. It cannot demonstrate that **there are no remaining faults**.

Software testing terms

Fault

Failure

Defect

Error



Software testing terms

Error:

- Error is **deviation from actual and expected value**.
- It represents mistake made by people.

Fault

- Fault is **incorrect step**, process or data definition in a computer program which causes the program to behave in an unintended or unanticipated manner.
- It is **the result of the error**.

Bug

- Bug is a fault in the program which causes the program to behave in an unintended or unanticipated manner.
- It is an evidence of fault in the program.

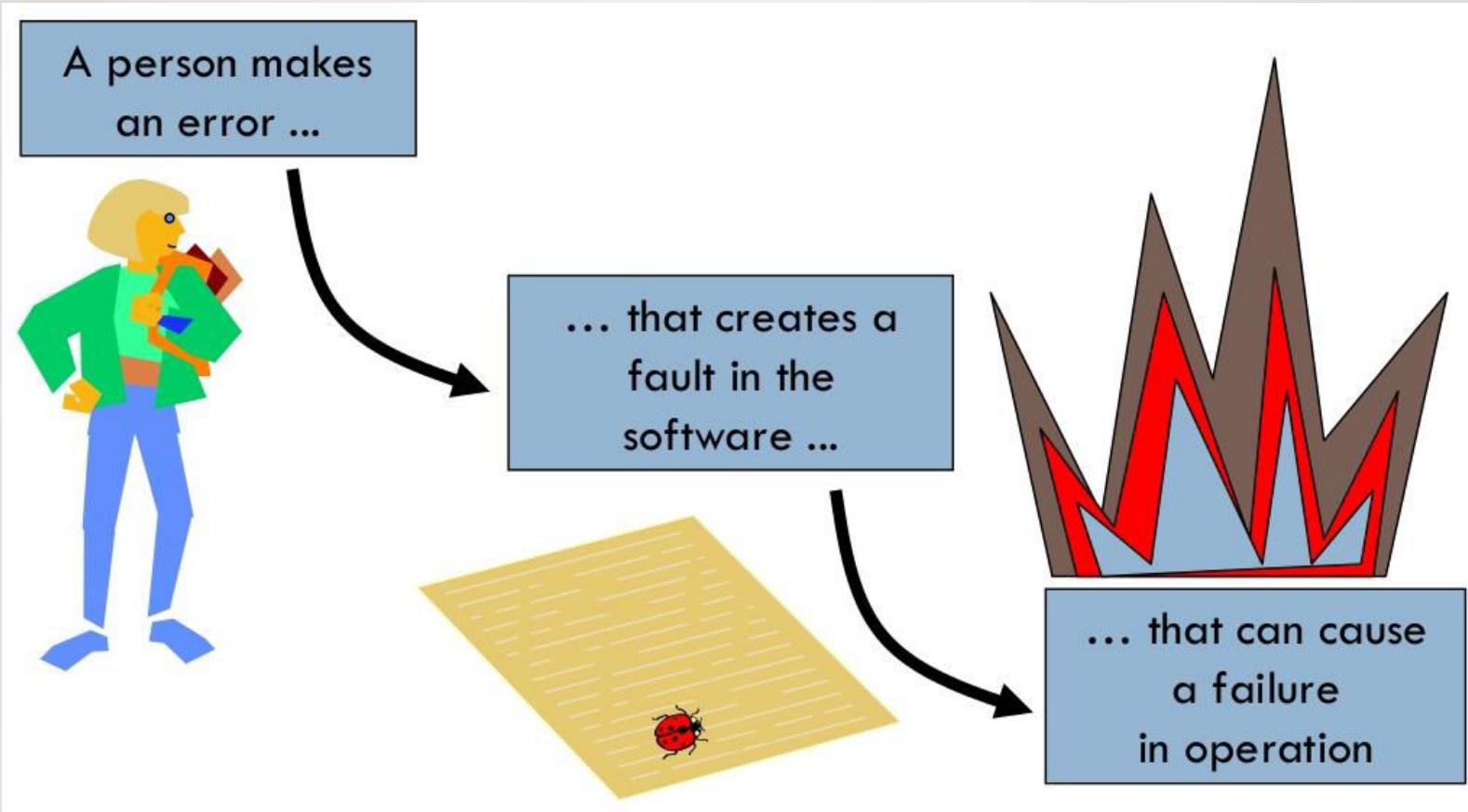
Failure

- Failure is the inability of a system or a component to perform its required functions within specified performance requirements.
- Failure occurs when fault executes.

Defect

- A defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results.
- A defect is said to be detected when a failure is observed.

Software testing terms



Error / Mistake	Defect / Bug/ Fault	Failure
Found by Developer	Found by Tester	Found by Customer

Benefits of Software Testing

What are the benefits of Software Testing?

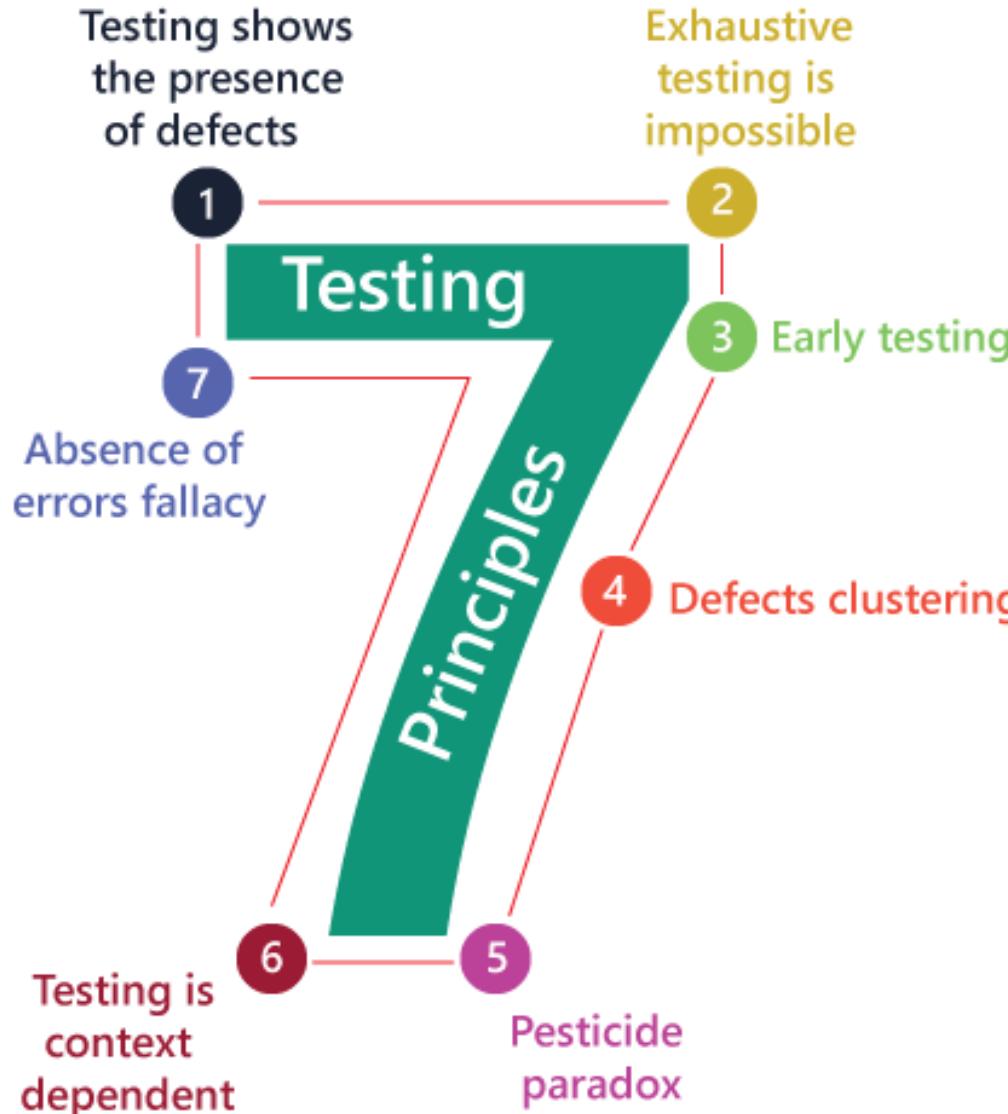
Cost-Effective: It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.

Security: It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.

Product quality: It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.

Customer Satisfaction: The main aim of any product is to give satisfaction to their customers. UI/UX Testing ensures the best user experience.

Software Testing Principles



Software Testing Principles

Testing shows presence of defects:

The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing **talks about the presence of defects and doesn't talk about the absence of defects**. Software testing can ensure that defects are present but it can not prove that software is defects free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not removes all defects.

Exhaustive testing is not possible:

It is the process of testing the functionality of a software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the **software can never test at every test cases**. It can test only some test cases and assume that software is correct and it will produce the correct output in every test cases. If the software will test every test cases then it will take more cost, effort, etc. and which is impractical.

Early Testing:

To find the defect in the software, early test activity shall be started. The defect detected in early phases of SDLC will very less expensive. For better performance of software, software testing will start at initial phase i.e. testing will perform at the requirement analysis phase.

Software Testing Principles

Defect clustering:

In a project, a small number of the module can contain most of the defects. **Pareto Principle** to software testing state that **80% of software defect comes from 20% of modules.**

Pesticide paradox:

Repeating the same test cases again and again will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

Testing is context dependent:

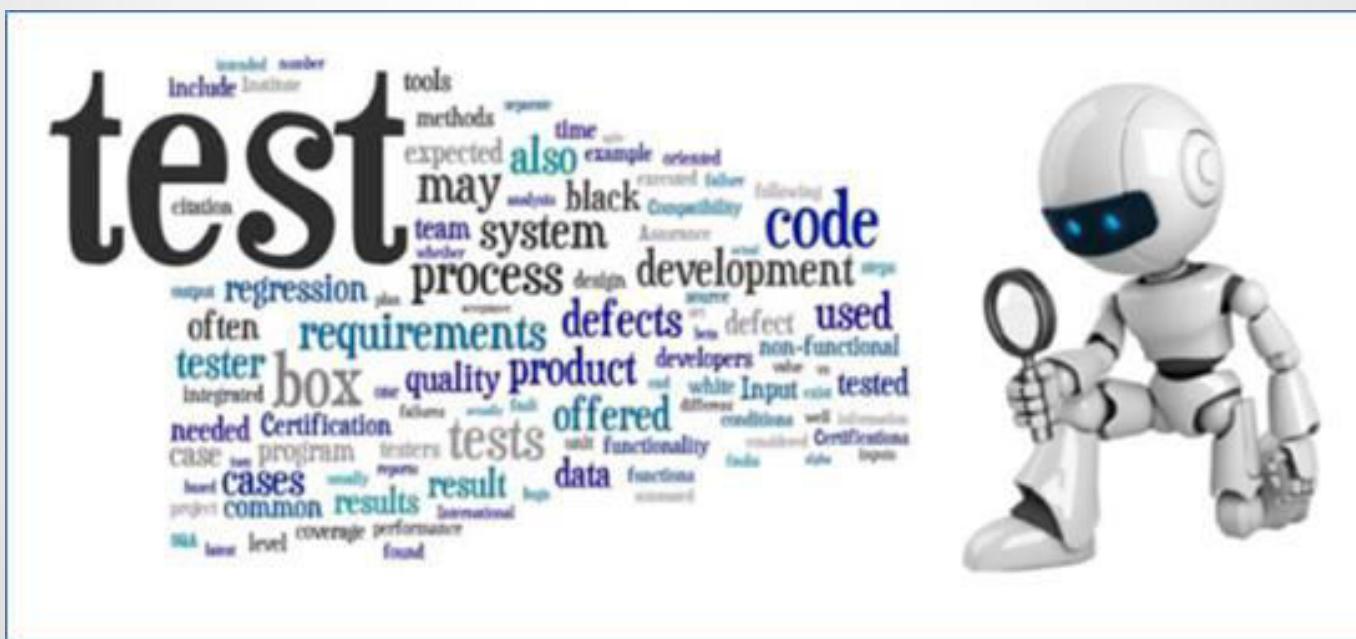
Testing approach depends on context of software developed. **Different types of software need to perform different types of testing.** For example, The testing of the e-commerce site is different from the testing of the Android application.

Absence of errors fallacy:

If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. **It is not only necessary that software is 99% bug-free but it also mandatory to fulfill all the customer requirements.**

Testing Guidelines

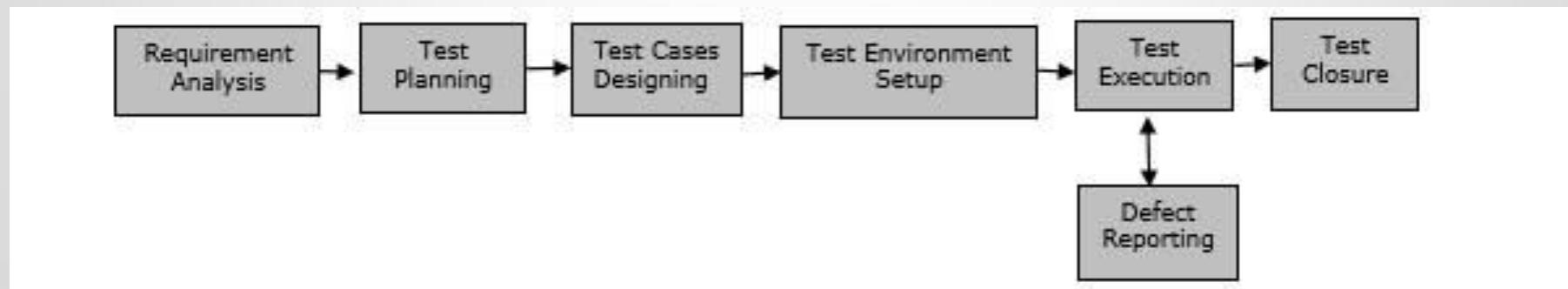
- Development team should avoid testing the software
- Testing must be done with unexpected and negative inputs
- Software can never be 100% bug-free after making a number of test cases
- Start as early as possible
- The time available is limited
- Inspecting test results properly
- Validating assumptions



Software Testing Life Cycle

STLC stands for **Software Testing Life Cycle**. STLC is a sequence of different activities performed by the testing team to **ensure the quality of the software or the product**.

- STLC is an integral part of Software Development Life Cycle (SDLC).
But, STLC deals only with the testing phases.
- STLC starts as soon as requirements are defined or SRD (Software Requirement Document) is shared by stakeholders.
- STLC provides a step-by-step process to ensure quality software.



Software Testing Life Cycle

There are 6 major phases of STLC –

- 1. Requirement Analysis** – When the SRD is ready and shared with the stakeholders, the testing team starts high level analysis concerning the AUT (Application under Test).
- 2. Test Planning** – Test Team plans the strategy and approach.
- 3. Test Case Designing/Developing** – Develop the test cases based on scope and criteria's.
- 4. Test Environment Setup** – When integrated environment is ready to validate the product.
- 5. Test Execution** – Real-time validation of product and finding bugs.
- 6. Test Closure** – Once testing is completed, matrix, reports, results are documented.

Testing process

Component testing

- Testing of individual program components;
- Usually the responsibility of the component developer (except sometimes for critical systems);
- Tests are derived from the developer's experience.

System testing

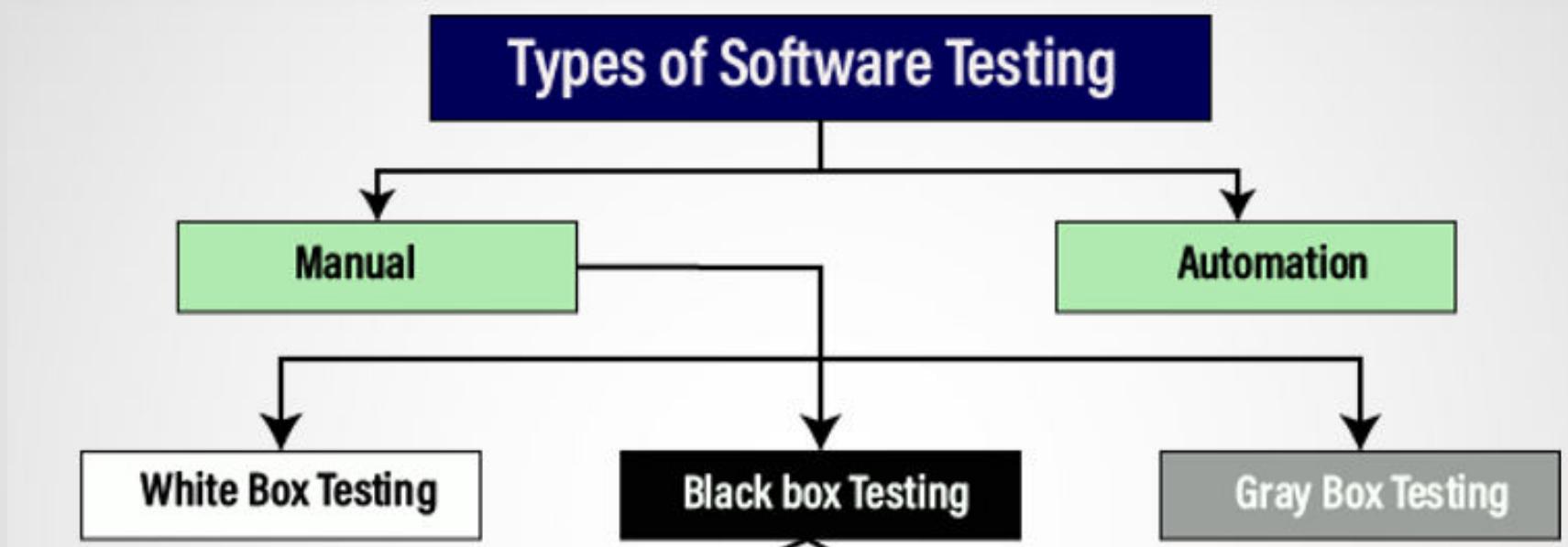
- Testing of groups of components integrated to create a system or subsystem;
- The responsibility of an independent testing team;
- Tests are based on a system specification



Software developer

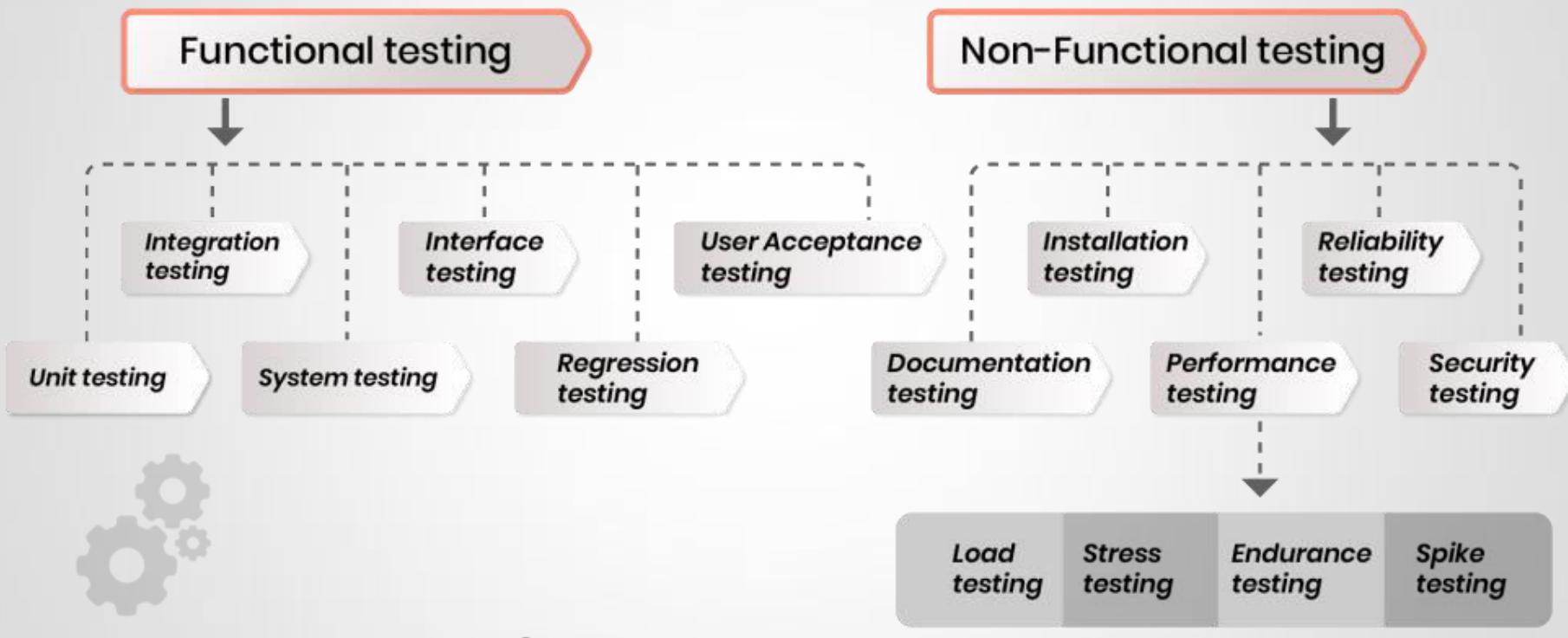
Independent testing team

Software Testing



Software Testing

TYPES OF SOFTWARE TESTING



Manual testing

Manual testing

The process of checking the functionality of an application as per the customer needs **without taking any help of automation tools** is known as manual testing.

While performing the manual testing on any application, we do not need any specific knowledge of any testing tool, rather than have a proper understanding of the product so we can easily prepare the test document.

Manual testing can be further divided into three types of testing, which are as follows:

- **White box testing**
- **Black box testing**
- **Gray box testing**

Manual testing

Advantages of Manual Testing

- It does not require programming knowledge while using the Black box method.
- It is used to test dynamically changing GUI designs.
- Tester **interacts with software as a real user** so that they are able to discover usability and user interface issues.
- It is **cost-effective**.
- Easy to learn for new testers.

Disadvantages of Manual Testing

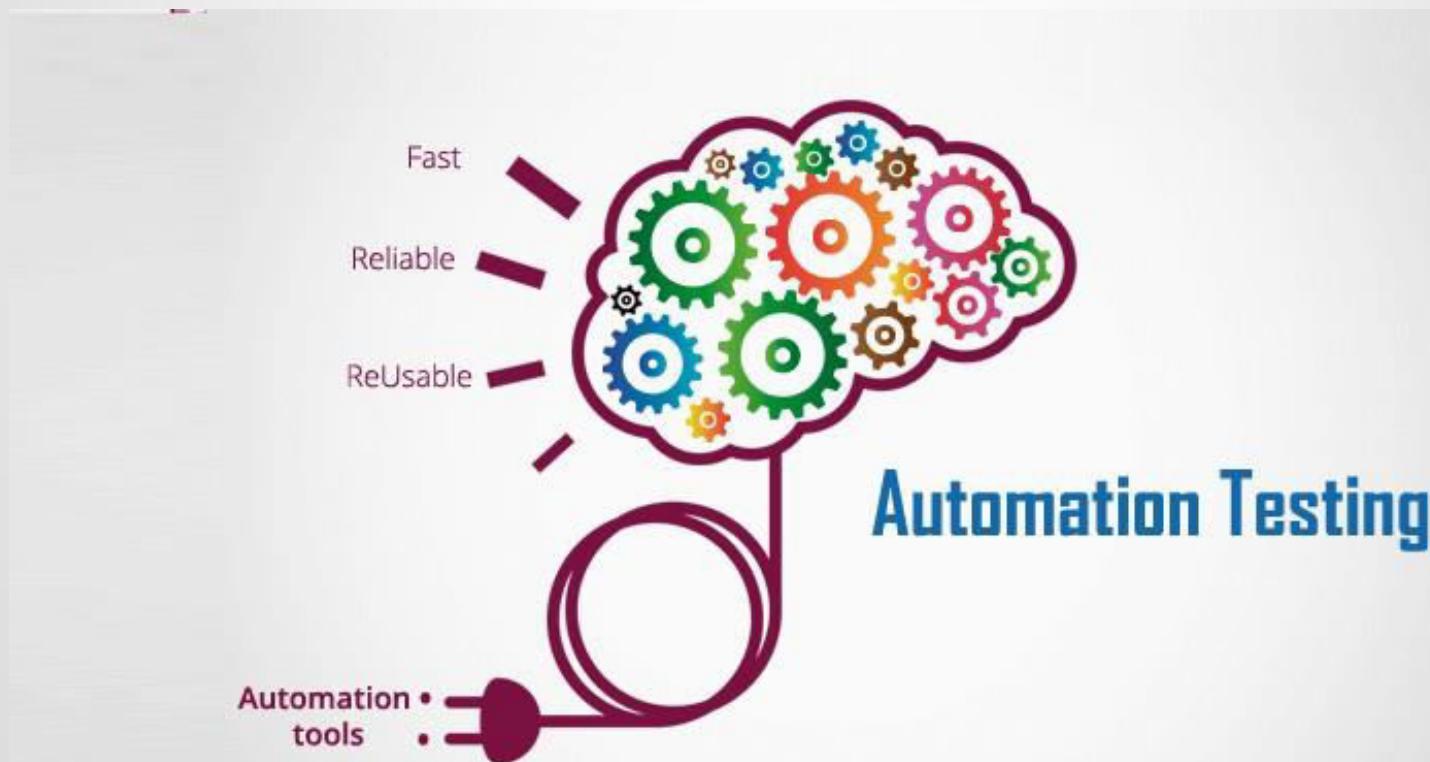
- It requires a large number of human resources.
- It is **very time-consuming**.
- Tester develops test cases based on their skills and experience. There **is no evidence that they have covered all functions or not**.
- Test cases **cannot be used again**. Need to develop separate test cases for each new software.
- It does not provide testing on all aspects of testing.
- Since two teams work together, sometimes it is difficult to understand each other's motives, **it can mislead the process**.

Manual testing tools

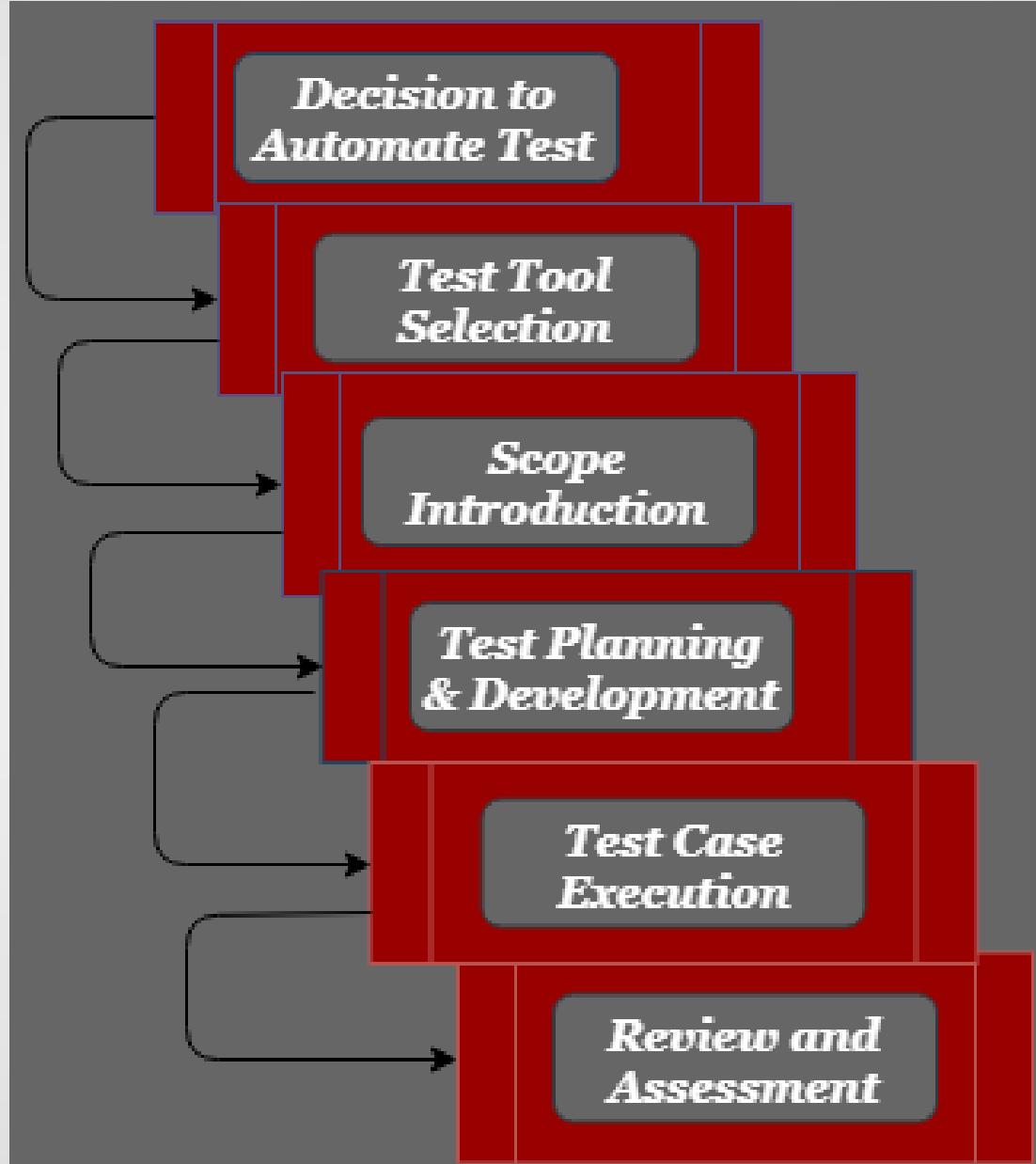


Automation testing

Automation testing is a process of **converting any manual test cases into the test scripts** with the help of **automation tools**, or any programming language is known as automation testing.



Life cycle of Automation Testing



Automation Testing

Advantages of Automation Testing

- Automation testing takes **less time** than manual testing.
- Automation Testing provides **re-usability of test cases** on testing of different versions of the same software.
- Automation testing is **reliable as it eliminates hidden errors** by executing test cases again in the same way.
- It does not require many human resources, instead of writing test cases and testing them manually, they need an automation testing engineer to run them.

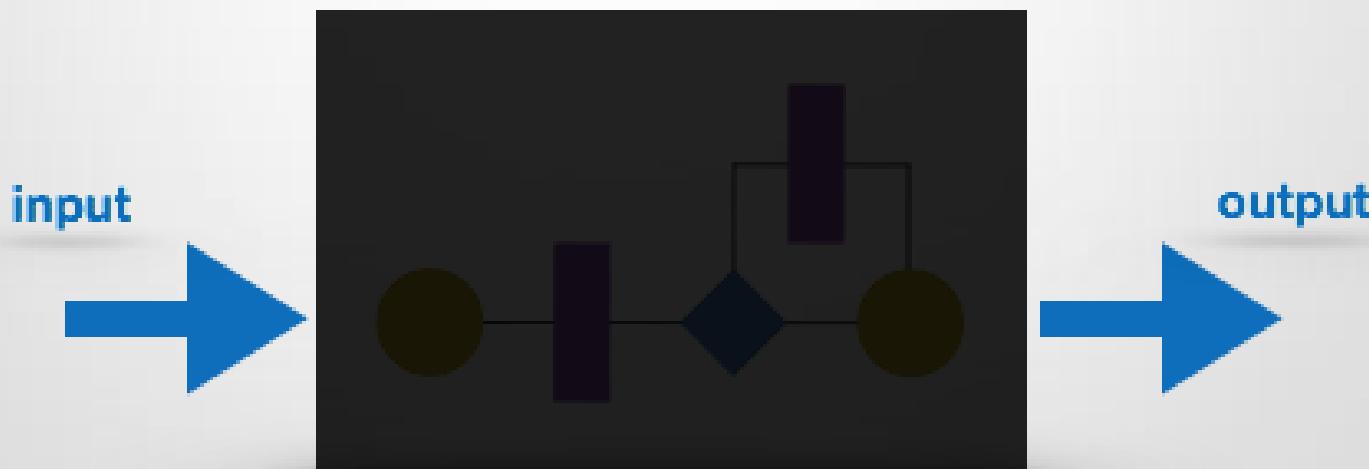
Disadvantages of Automation Testing

- Automation Testing **requires high-level skilled testers**.
- It requires **high-quality testing tools**.
- When it encounters an unsuccessful test case, the analysis of the whole event is complicated.
- Test **maintenance is expensive** because high fee license testing equipment is necessary.
- Debugging is mandatory if a less effective error has not been solved, it can lead to fatal results.

Black box testing

Black box Testing

- **Black box testing** is a technique of software testing which examines the functionality of software **without peering into its internal structure or coding**.
- Black box testing is a method of software testing that **examines the functionality of an application**
- This method of test can be applied to virtually every level of software
- The tester is oblivious to the system architecture and does not have access to the source code



Black box testing

Generic steps of black box testing

- ❖ The black box test is based on the specification of requirements, so it is examined in the beginning.
- ❖ In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
- ❖ In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
- ❖ The fourth phase includes the execution of all test cases.
- ❖ In the fifth step, the tester compares the expected output against the actual output.
- ❖ In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

Black box testing

Black-box testing techniques:

Decision Table Technique	Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form . It is appropriate for the functions that have a logical relationship between two and more than two inputs.
Boundary Value Technique	Boundary Value Technique is used to test boundary values , boundary values are those that contain the upper and lower limit of a variable. It tests, while entering boundary value whether the software is producing correct output or not.
State Transition Technique	State Transition Technique is used to capture the behavior of the software application when different input values are given to the same function . This applies to those types of applications that provide the specific number of attempts to access the application.
All-pair Testing Technique	All-pair testing Technique is used to test all the possible discrete combinations of values . This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc.

Black box testing

Black-box testing techniques:

Cause-Effect Technique	Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result. It is based on a collection of requirements.
Equivalence Partitioning Technique	Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values , and it is mandatory that all partitions must exhibit the same behavior.
Error Guessing Technique	Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software.
Use Case Technique	Use case Technique used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

Black box testing

Advantages of Black Box Testing

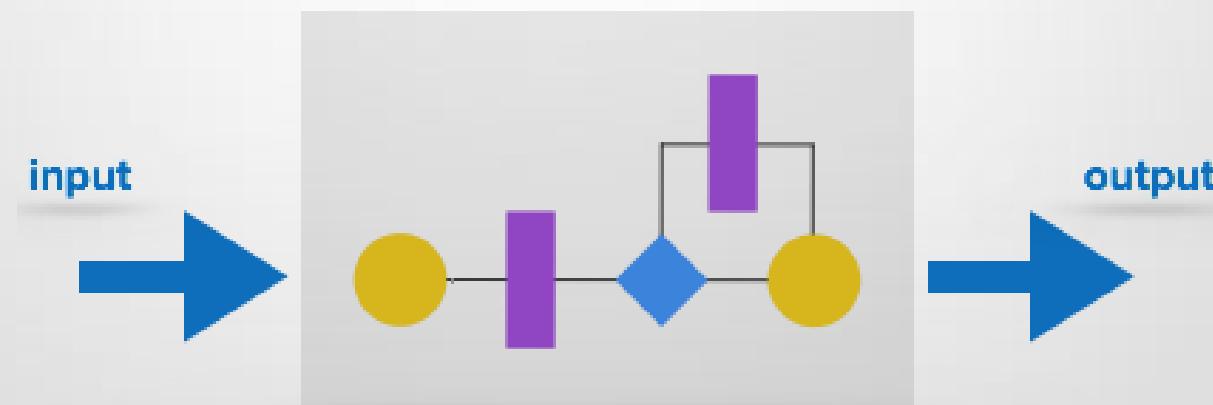
- Tester can be non-technical.
- Used to verify contradictions in actual system and the specifications.
- Test cases can be designed as soon as the functional specifications are complete

Disadvantages of Black Box Testing

- The test inputs needs to be from large sample space.
- It is difficult to identify all possible inputs in limited testing time. So writing test cases is slow and difficult
- Chances of having unidentified paths during this testing

White Box Testing

- **White box testing** is a method of testing software that **tests internal structures or working of an application**
- In white-box testing an internal perspective of the system , as well as programming skills, are used to design test cases
- It is also known as **clear box testing, glass box testing, transparent box testing, and structural testing**
- White box testing is the detailed investigation of internal logic and structure of the code
- In order to perform white box testing of an application , the tester needs to possess knowledge of the **internal working of the code**

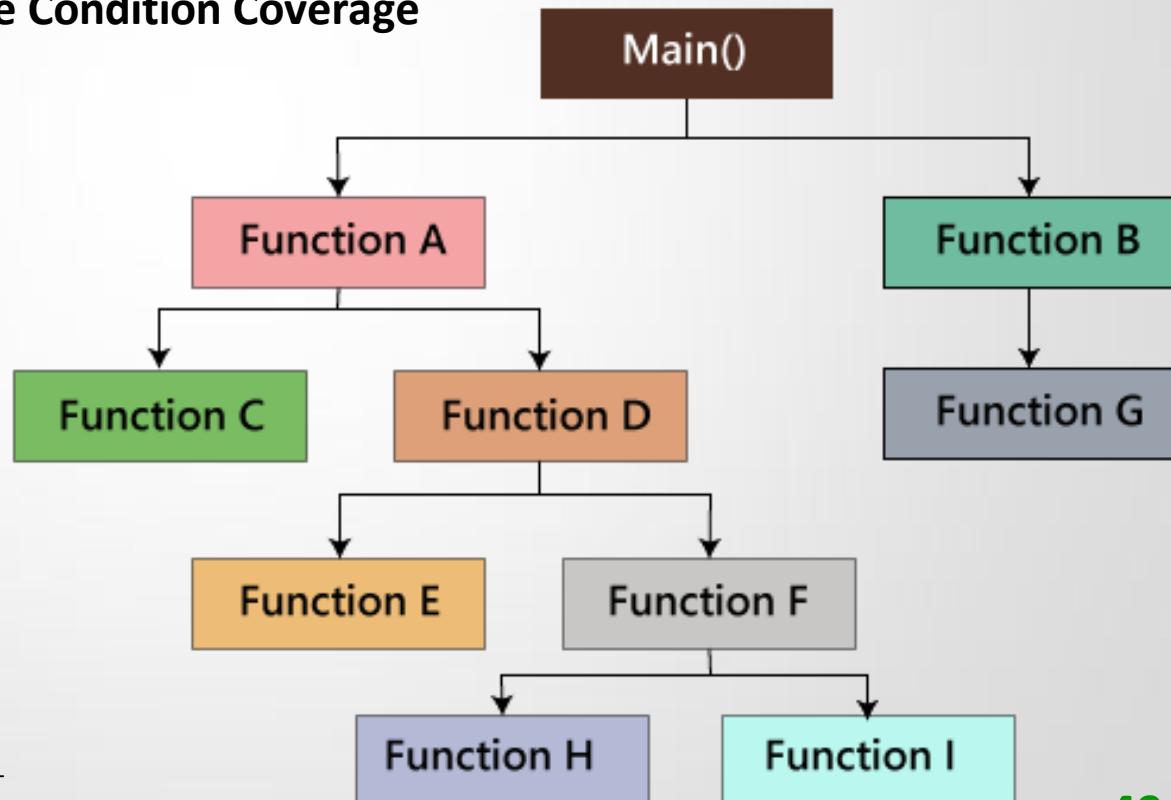
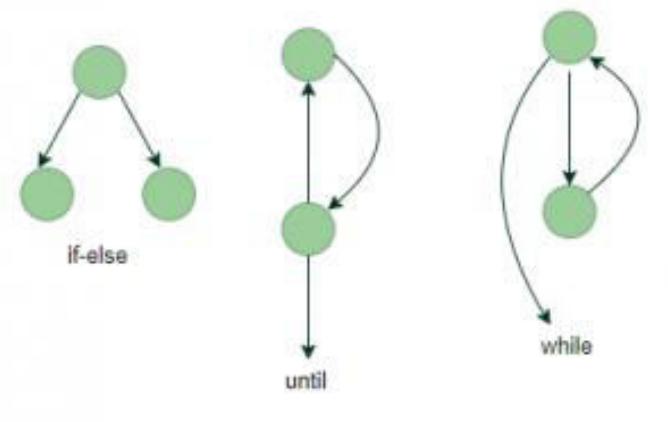


White Box Testing

White Box Testing techniques:

The white box testing contains various parts, which are as follows:

- Statement coverage
- Testing based on the memory (size) perspective
- Condition testing : **Multiple Condition Coverage**
- Basis Path test
- Flow graph notation
- Loop Testing



White Box Testing

Advantages:

- 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:

- 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.

Black Box Testing vs White Box Testing

BLACK BOX TESTING	WHITE BOX TESTING
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure r the code or the program of the software.
It is mostly done by software testers .	It is mostly done by software developers .
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred as outer or external software testing.	It is the inner or the internal software testing.
It is functional test of the software.	It is structural test of the software.
This testing can be initiated on the basis of requirement specifications document.	This type of testing of software is started after detail design document.

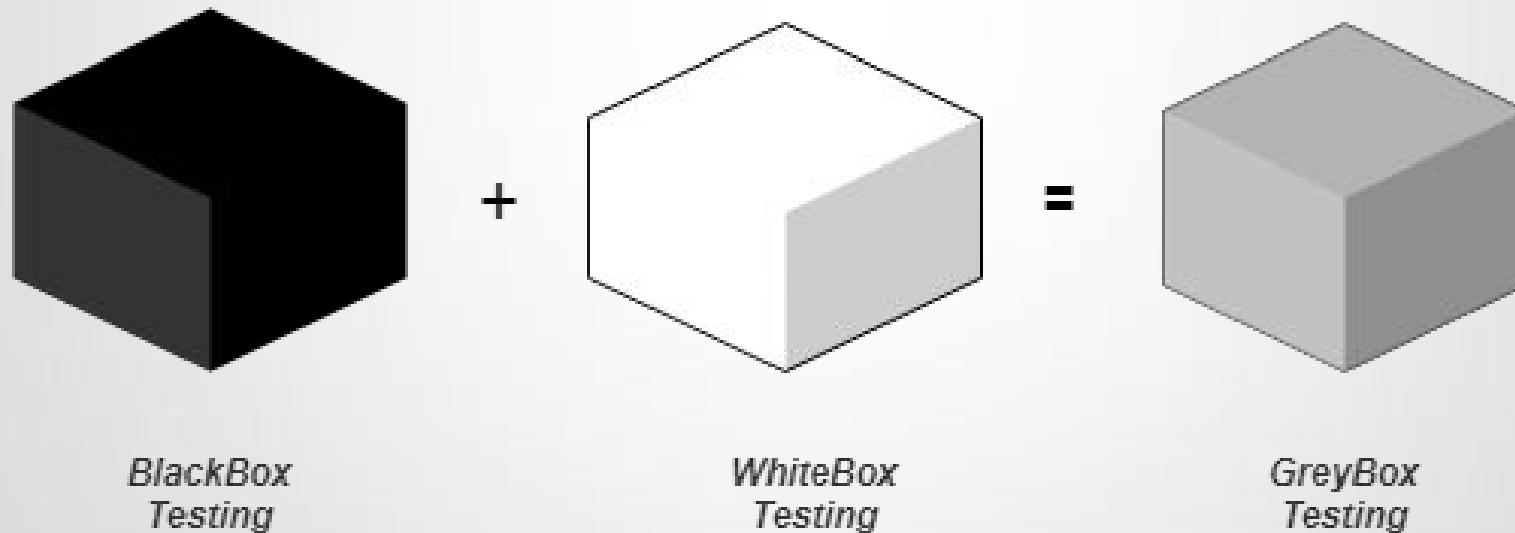
Black Box Testing vs White Box Testing (cont.)

BLACK BOX TESTING	WHITE BOX TESTING
No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing .	It is also called as clear box testing .
It is least time consuming.	It is most time consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.

GreyBox Testing

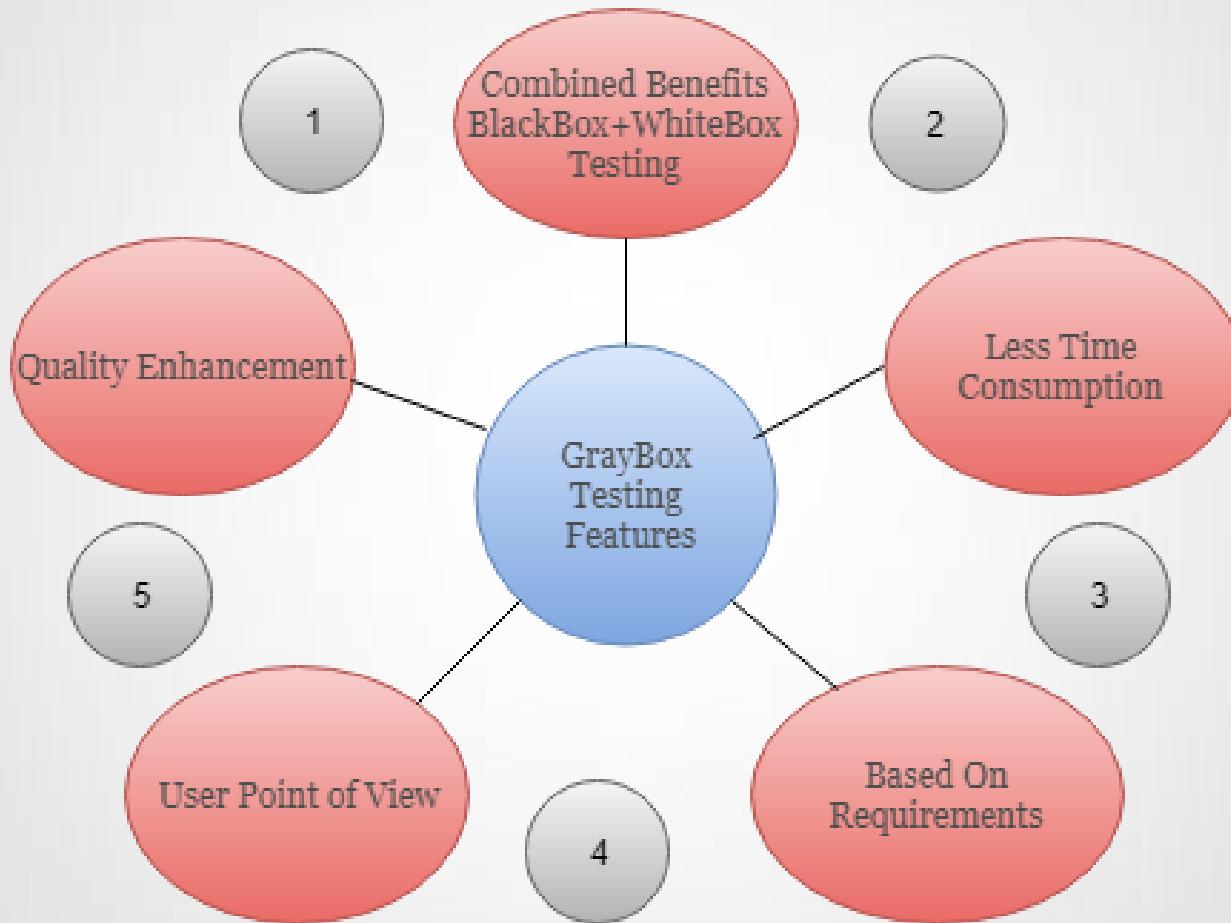
Grey-box testing is a software testing method to test the software application with partial knowledge of the internal working structure.

It is a **combination of black box and white box testing** because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing.



GreyBox Testing

Why GreyBox testing?



GreyBox Testing

Why GreyBox testing?

Reasons for GreyBox testing are as follows

- ❖ It provides combined benefits of both Blackbox testing and WhiteBox testing.
- ❖ It includes the input values of **both developers and testers** at the same time to **improve the overall quality** of the product.
- ❖ It **reduces time consumption** of long process of functional and non-functional testing.
- ❖ It **gives sufficient time** to the developer to fix the product defects.
- ❖ It includes user point of view rather than designer or tester point of view.
- ❖ It involves examination of requirements and determination of specifications by user point of view deeply.

Gray Box Testing Techniques:

- ❖ Matrix Testing
- ❖ Pattern Testing
- ❖ Orthogonal Array Testing
- ❖ Regression Testing

GreyBox Testing

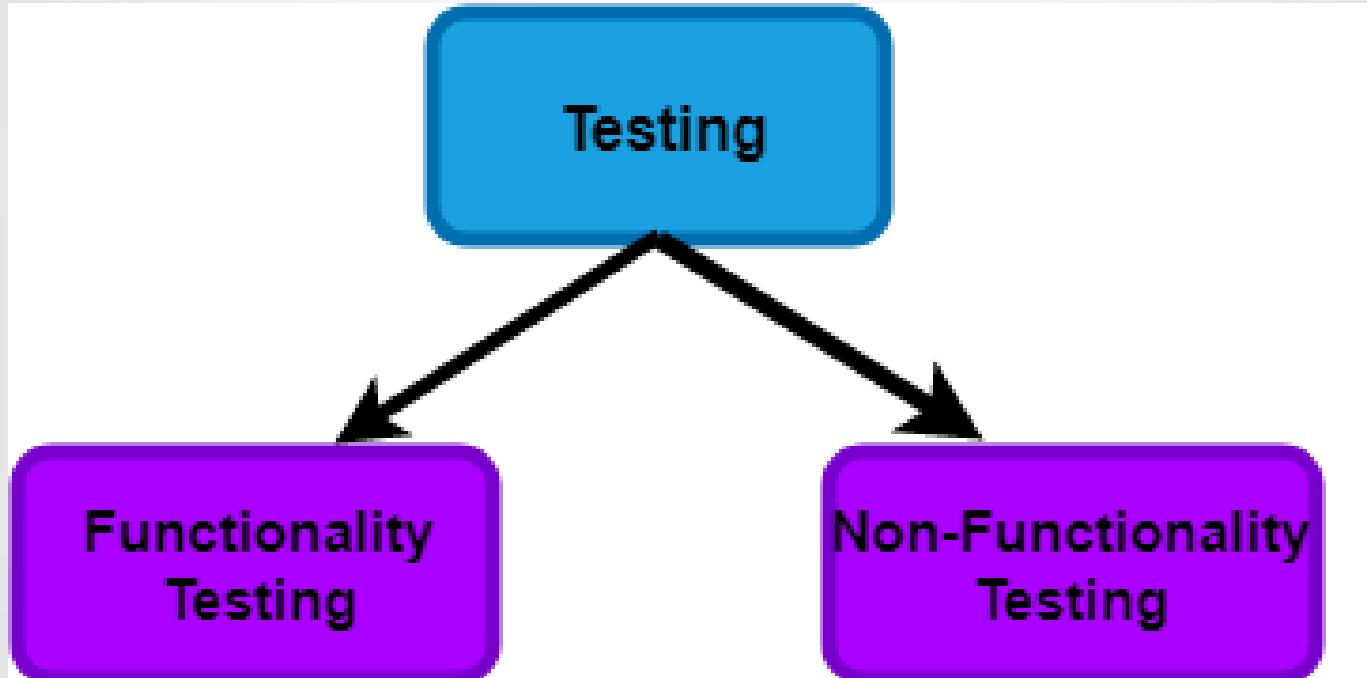
Advantages of Gray Box Testing:

- ❖ Gray box testing is mostly done by the **user perspective**.
- ❖ Testers are **not required to have high programming skills** for this testing.
- ❖ In gray box testing, developers have more **time for defect fixing**.
- ❖ By doing gray box testing, benefits of both black box and white box testing is obtained.
- ❖ Gray box testing is **unbiased**. It avoids conflicts between a tester and a developer.

Disadvantages of gray box testing:

- ❖ Defect association is difficult when gray testing is performed for **distributed systems**.
- ❖ Limited access to internal structure leads to limited access for code **path traversal**.
- ❖ Because source code cannot be accessed, doing complete white box testing is not possible.
- ❖ Gray box testing is not suitable for **algorithm testing**.

Testing Approaches



Testing Approaches

Functional Testing:

It is a type of software testing which is used to **verify the functionality of the software application**, whether the function is **working according to the requirement specification**.

- Tester does verification of the requirement specification in the software application.
- After analysis, the requirement specification tester will make a plan.
- After planning the tests, the tester will design the test case.
- After designing the test, case tester will make a document of the traceability matrix.
- The tester will execute the test case design.
- Analysis of the coverage to examine the covered testing area of the application.
- Defect management should do to manage defect resolving.

Testing Approaches

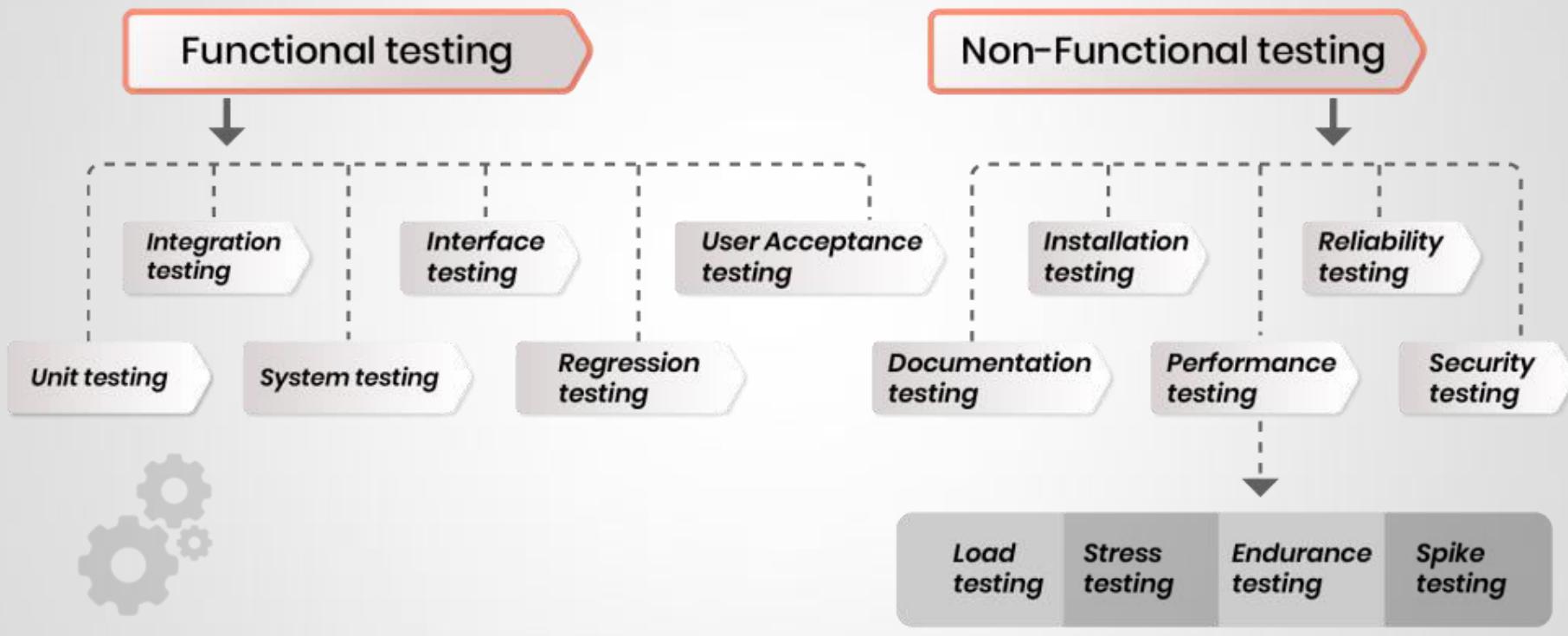
Non-Functional Testing

Non-functional testing is a type of software testing to test **non-functional parameters** such as reliability, load test, performance and accountability of the software. The primary purpose of non-functional testing is to test the reading speed of the software system as per non-functional parameters.



Software Testing

TYPES OF SOFTWARE TESTING



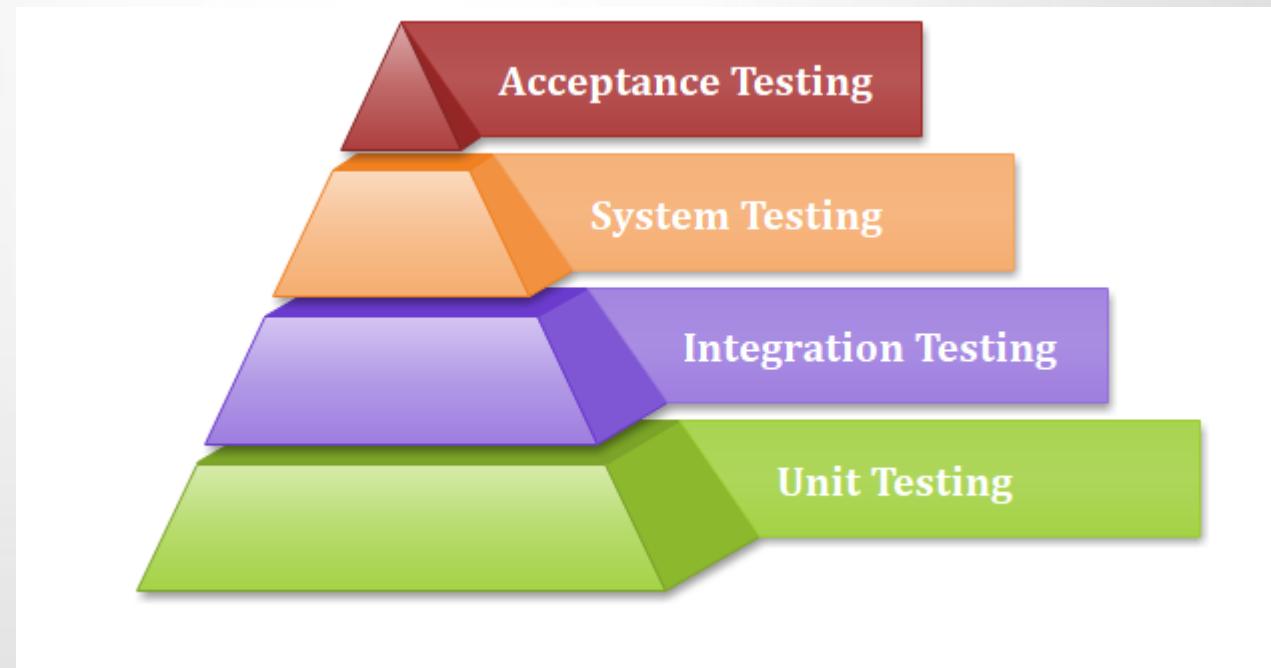
Unit Testing

Unit testing is a type of software testing, where the **individual unit or component of the software tested**. Unit testing, **examine the different part of the application**, by unit testing functional testing also done, because unit testing ensures each module is working correctly.

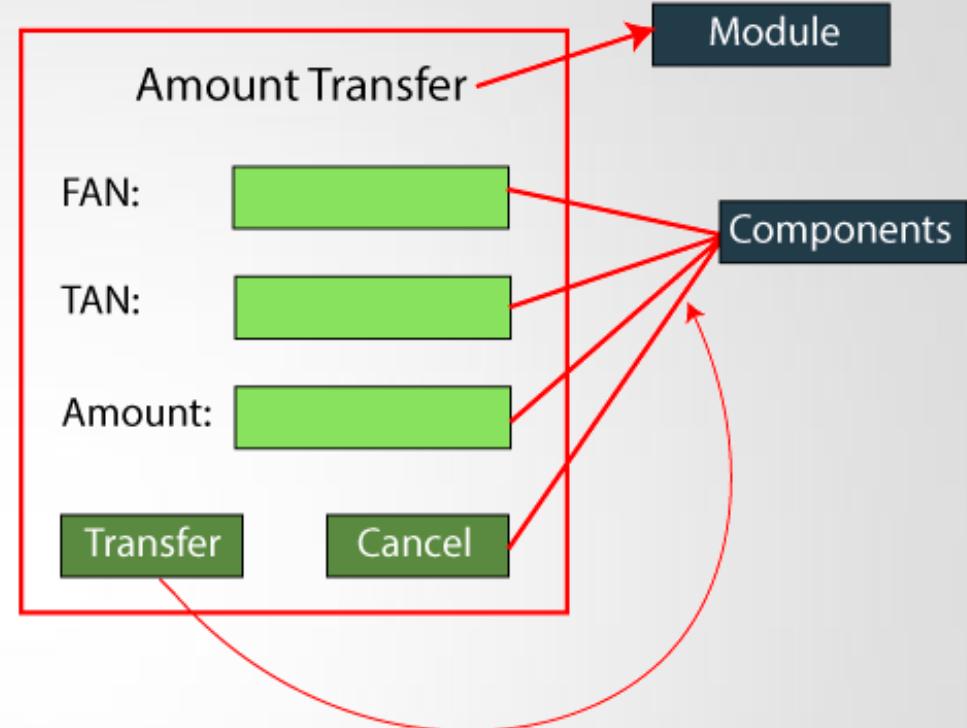
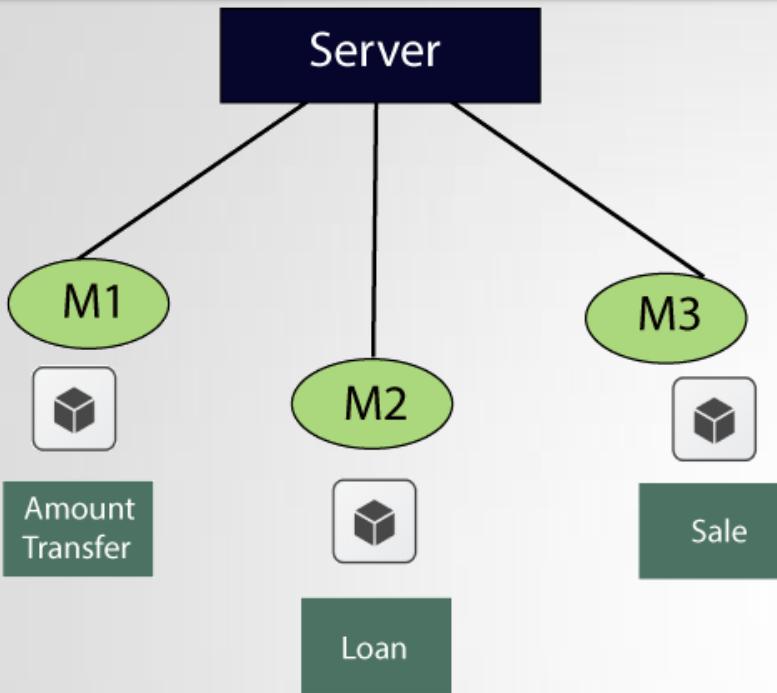
The developer does unit testing. Unit testing is done in the development phase of the application.

Unit Testing Tools

- NUnit
- JUnit
- PHPunit
- Parasoft Jtest
- EMMA



Unit Testing



Unit Testing Techniques:

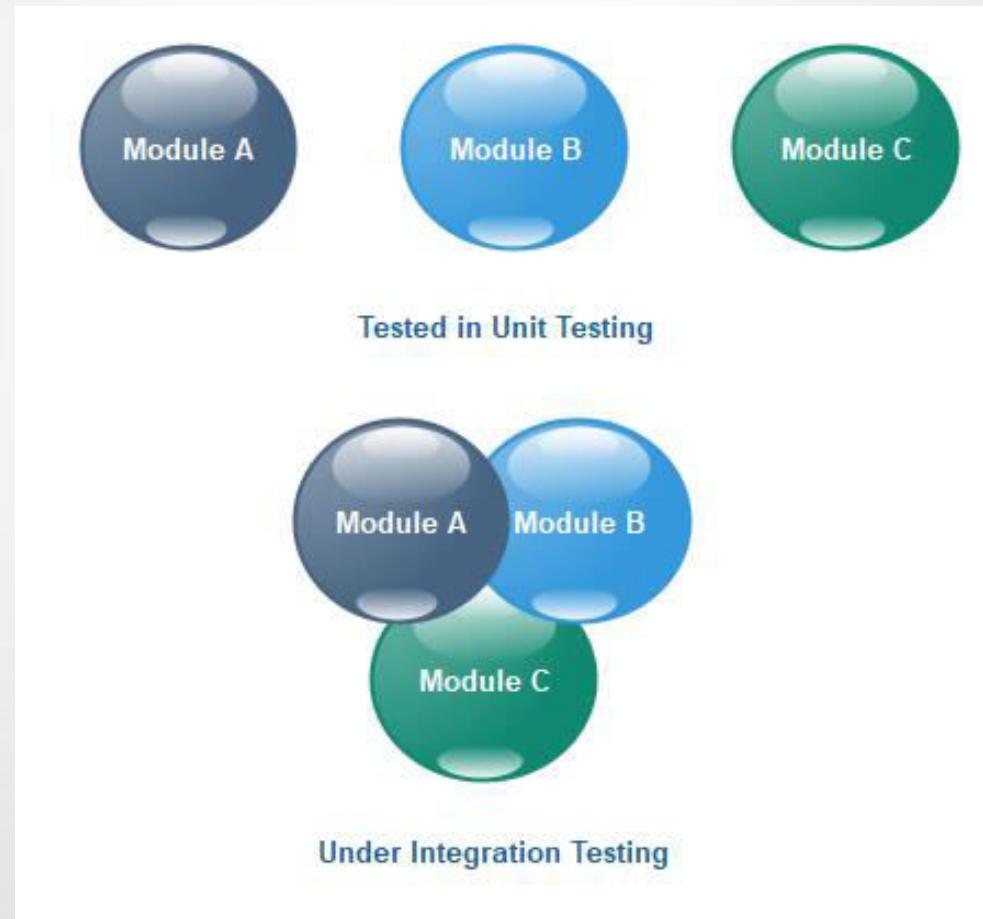
Unit testing uses all white box testing techniques as it uses the code of software application:

- **Data flow Testing**
- **Control Flow Testing**
- **Branch Coverage Testing**
- **Statement Coverage Testing**
- **Decision Coverage Testing**

Integration testing

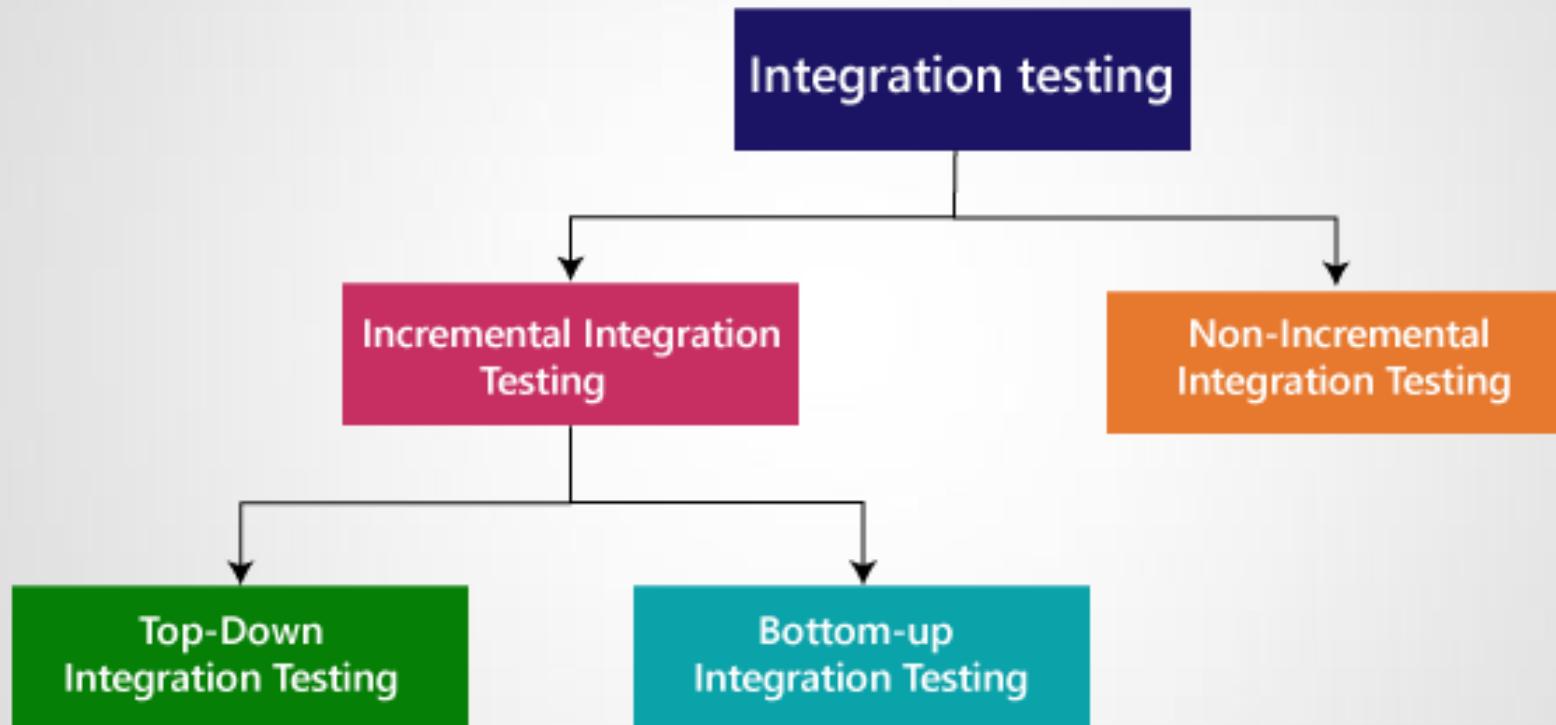
Integration testing

Integration testing is the second level of the software testing process **comes after unit testing**. In this testing, units or individual components of the software are **tested in a group**. The focus of the integration testing level is to **expose defects at the time of interaction between integrated components or units**.



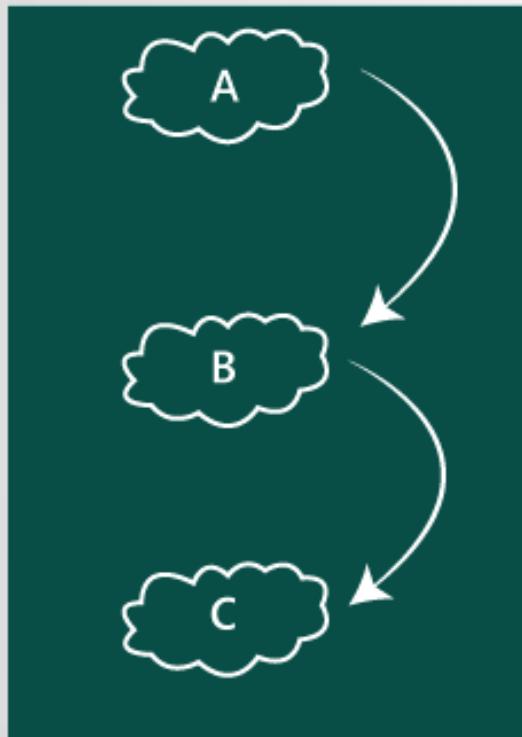
Integration testing

Types of Integration Testing

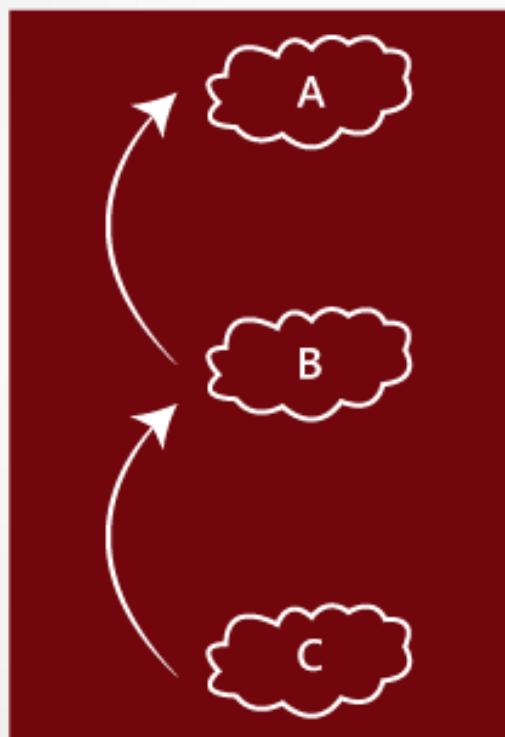


Integration testing

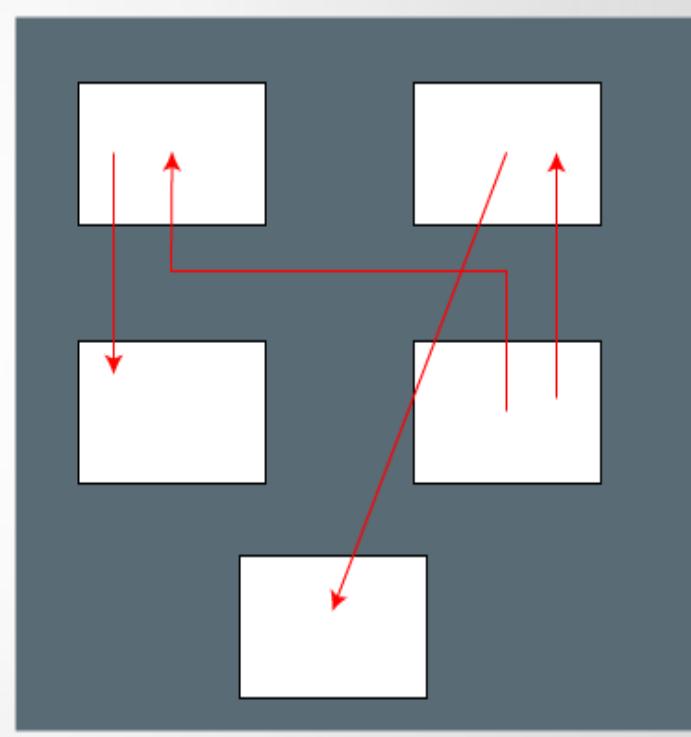
Top-Down Approach



Bottom-up Approach



Non-incremental

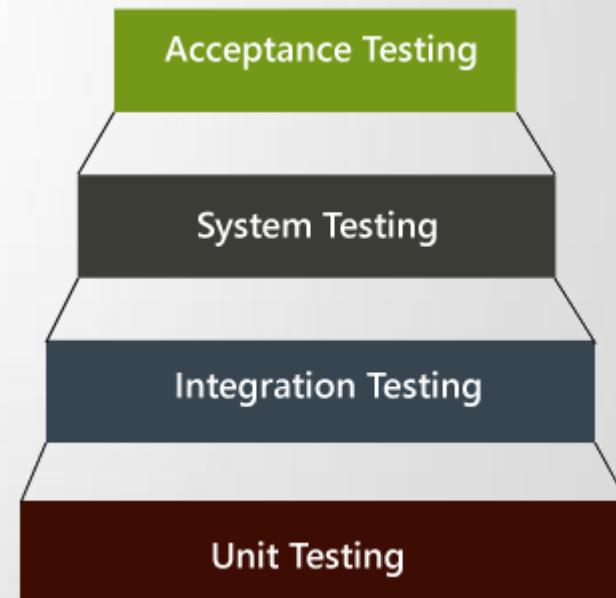


System Testing

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:

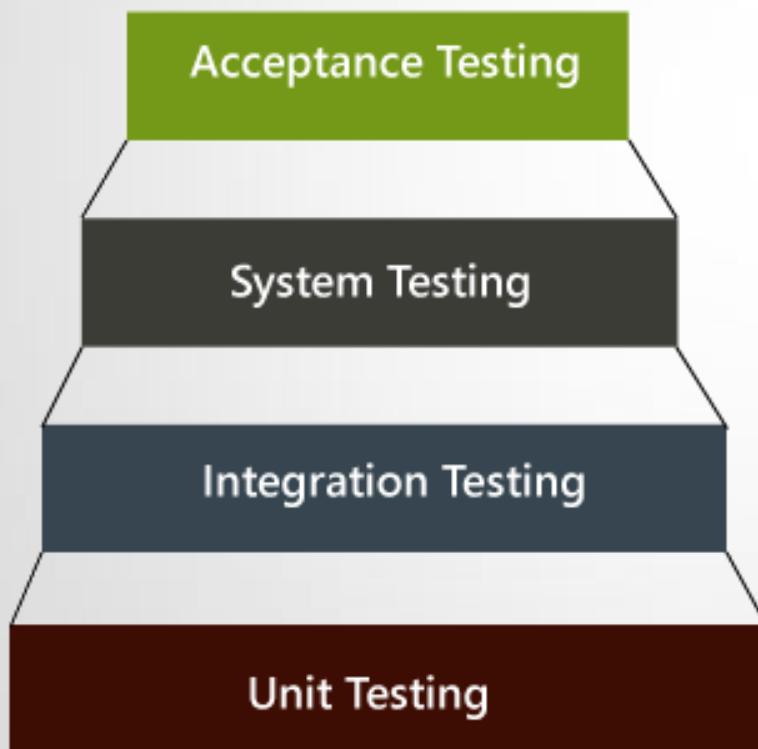
- **Functionality testing** - Tests all functionalities of the software against the requirement.
- **Performance testing** - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. Performance testing is done by means of load 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.



System Testing

System Testing

- The software is compiled as product and then **it is tested as a whole**.
- System Testing includes testing of a fully integrated software system.
- System testing is divided into more **than 50 types**.



Acceptance Testing : User testing

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.

Different types of testing

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.

Load Testing

Load testing is performed under system testing to clarify whether the system can work under real-time loads or not.

Stress testing

The stress testing is testing, which checks the behavior of an application by applying load greater than the desired load. Since it is non-functional testing, so we use this testing when the application is functionally stable.

Recovery Testing

Recovery testing of a system is performed under system testing to confirm reliability, trustworthiness, accountability of the system and all are lying on recouping skills of the system. It should be able to recover from all the possible system crashes successfully. In this testing, we will test the application to check how well it recovers from the crashes or disasters.

Different types of testing

Migration Testing

Migration testing is performed to ensure that if the system needs to be modified in new infrastructure so it should be modified without any issue.

Usability Testing

The purpose of this testing to make sure that the system is well familiar with the user and it meets its objective for what it supposed to do.

Database Testing: Database testing is a type of testing which **checks the schema, tables, triggers, etc. of the database under test.** Database testing may involve creating complex queries to load/stress test the database and check its responsiveness. It checks the data integrity and consistency

Release testing the process of testing a particular release of a system that is intended for use outside of the development team. Release testing has a broad focus, since the entire functionality of the release is under test. The tests included in release testing is strongly dependent on the software itself.

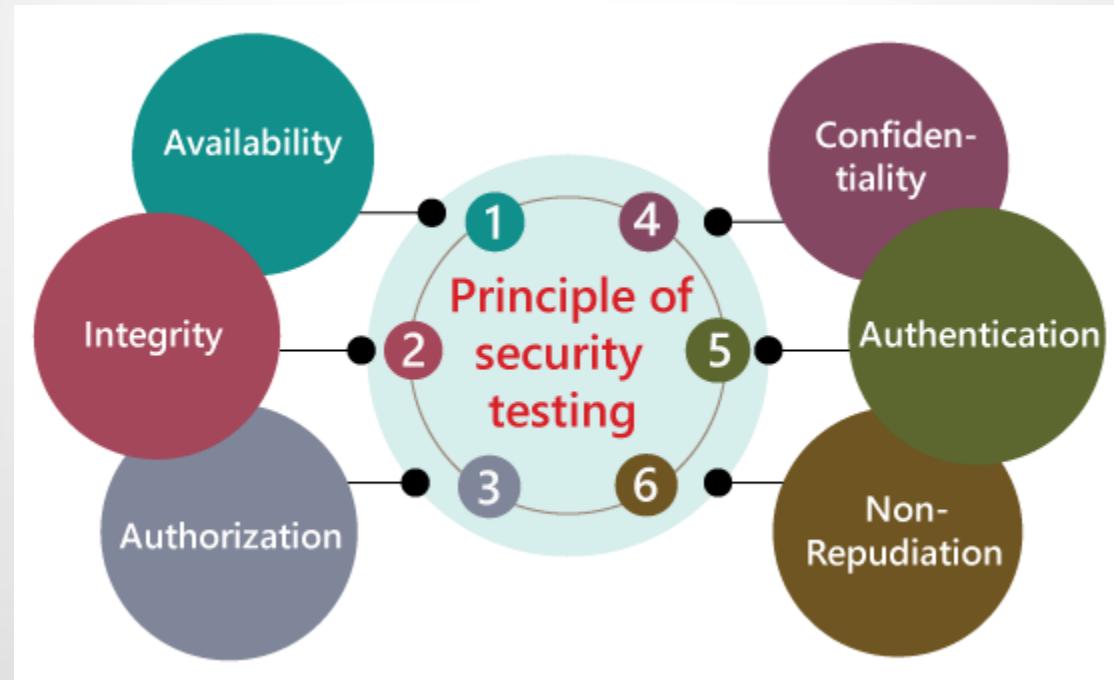
- Requirements-based testing
- Scenario testing

Different types of testing

Security testing

Security testing is an integral part of software testing, which is used to **discover the weaknesses, risks, or threats** in the software application and also help us to stop the nasty attack from the outsiders and make sure the security of our software applications.

The primary objective of security testing is to **find all the potential ambiguities and vulnerabilities** of the application so that the software does not stop working. If we perform security testing, then it helps us to identify all the possible security threats and also help the programmer to fix those errors.

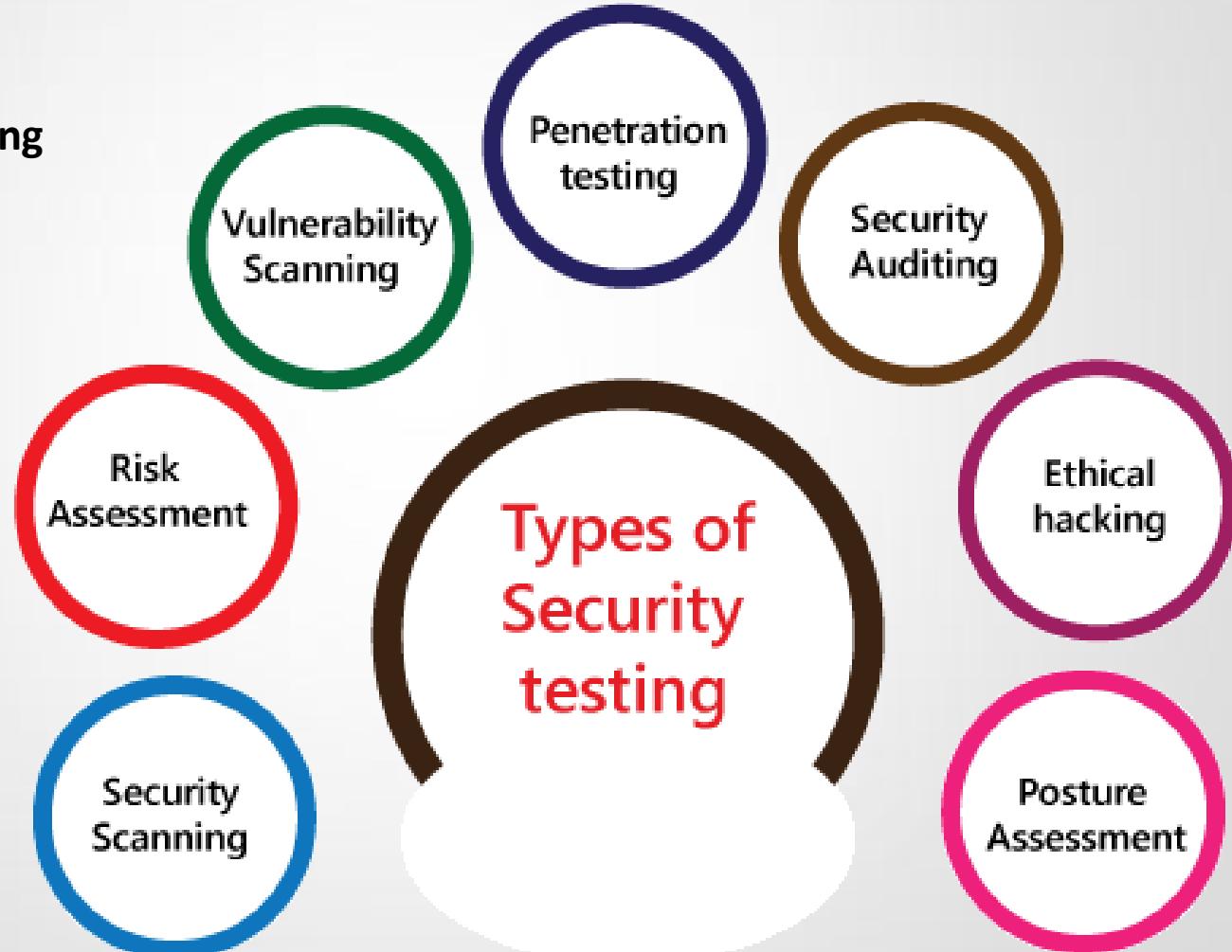


Security testing

Types of Security testing

As per Open Source Security Testing techniques, we have different types of security testing which as follows:

- ❖ Security Scanning
- ❖ Risk Assessment
- ❖ Vulnerability Scanning
- ❖ Penetration testing
- ❖ Security Auditing
- ❖ Ethical hacking
- ❖ Posture Assessment



Alpha testing vs Beta testing

Alpha Testing	Beta Testing
Alpha testing performed by a team of highly skilled testers who are usually the internal employee of the organization .	Beta testing performed by clients or end-users in a real-time environment, who is not an employee of the organization.
Reliability or security testing not performed in-depth in alpha testing.	Reliability, security, and robustness checked during beta testing.
Alpha testing involves both white box and black-box techniques.	Beta testing uses only black-box testing.
Long execution cycles maybe require for alpha testing.	Only a few weeks are required for the execution of beta testing.
Alpha testing performed before the launch of the product into the market.	At the time of software product marketing.
Alpha testing focuses on the product's quality before going to beta testing.	Beta testing concentrates on the quality of the product, but gathers users input on the product and ensures that the product is ready for real-time users .
Alpha testing performed nearly the end of the software development.	Beta testing is a final test before shipping a product to the customers.
Alpha testing is conducting in the presence of developers and the absence of end-users.	Beta testing reversed of alpha testing.

Re-testing Vs Regression Testing

Re-testing	Regression Testing
Re-testing is performed to ensure that the test cases that are failed in the final execution are passing after the defects fixed.	Regression Testing is done to confirm whether the code change has not affected the existing features .
Re-Testing works on defect fixes.	The purpose of regression testing is to ensure that the code changes adversely not affect the existing functionality.
Defect verification is the part of the Retesting.	Regression testing does not include defect verification
The priority of Retesting is higher than Regression Testing, so it is done before the Regression Testing.	Based on the project type and availability of resources, regression testing can be parallel to Retesting.
Re-Test is a planned Testing .	Regression testing is a generic Testing .
We cannot automate the test-cases for Retesting.	We can do automation for regression testing; manual testing could be expensive and time-consuming.
Re-testing is for failed test-cases .	Regression testing is for passed Test-cases .
Re-testing make sure that the original fault is corrected.	Regression testing checks for unexpected side effect.
Retesting executes defects with the same data and the same environment with different input with a new build.	Regression testing is when there is a modification or changes become mandatory in an existing project.

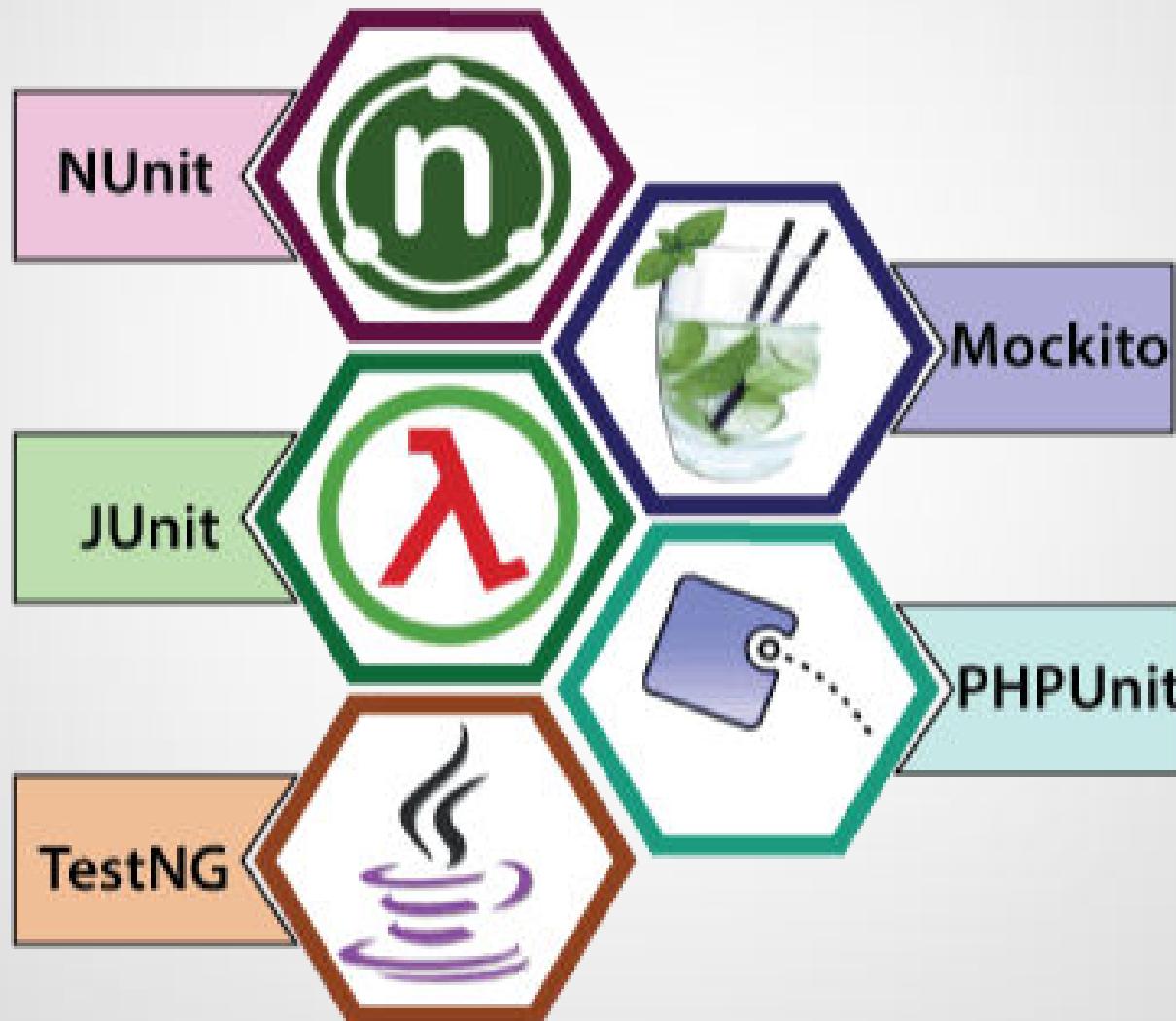
System testing vs Acceptance testing

System Testing	Acceptance Testing
System testing is performed to test end to end functionality of the software.	Acceptance testing is performed to test whether the software is conforming specified requirements and user requirements or not.
Only developers and testers can perform System testing.	It can be performed by testers, stakeholders and costumers.
It can be both non-functional and functional testing.	It can be only functional testing.
In System testing, we test the performance of the whole system.	In Acceptance testing, we test whether the system is conforming requirements or not.
System testing uses demo input values that are selected by the testing team.	Acceptance testing uses the actual real-time input values provided by the user.
System Testing is a combination of System part testing and Integration testing.	Acceptance Testing is a combination of alpha testing and beta testing.
It is performed before the Acceptance testing.	It is performed after the System testing.
System testing involves load and stress testing under non-functional testing.	Acceptance testing involves boundary value analysis, equivalence portioning and decision table under functional testing.
The defects found in system testing are considered to be fixed.	The defects found in acceptance testing are considered as product failure.

Testing Tools



Unit Testing Tools



Testing Tools

Integration Testing Tool



FitNesse



Protractor



Citrus

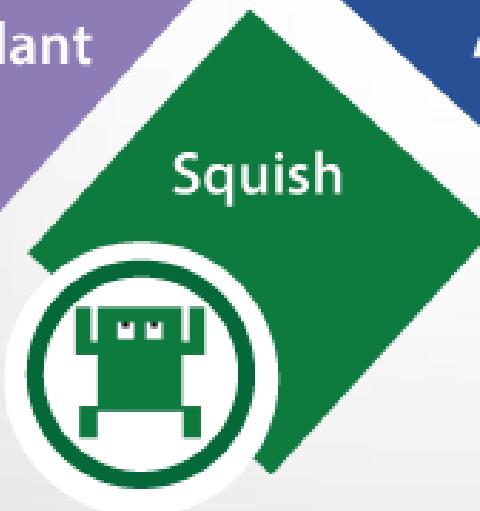
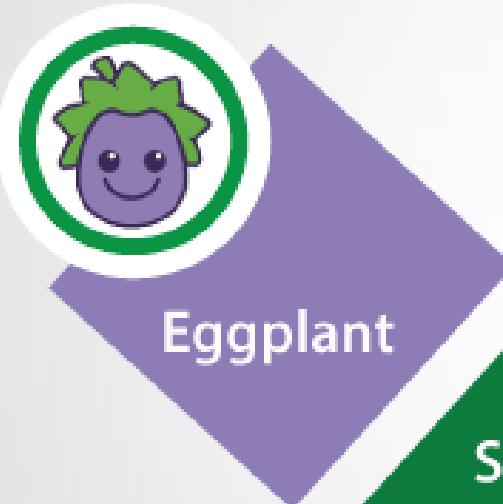
Rational software

Rational
Integration
tester



TESSY

GUI Testing Tools



Testing Tools

Performance (Load) Testing Tools



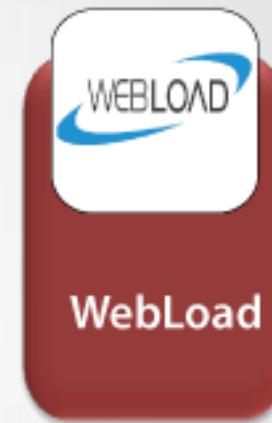
Apache
JMeter



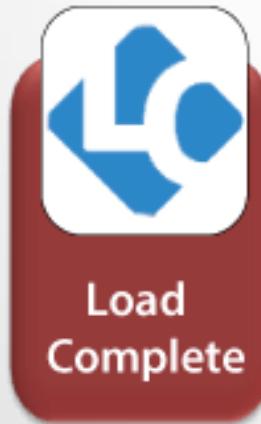
LoadNinja



LoadRunner



WebLoad



Load
Complete



NeoLoad



LoadView

Testing Tools

Cross-Browser Testing Tools



LambdaTest



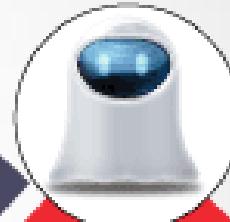
SauceLabs



CrossBrowser
Testing



BrowserStack



GhostLab



Browsera

Testing Tools

Testdroid



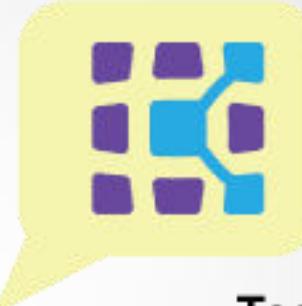
Calabash



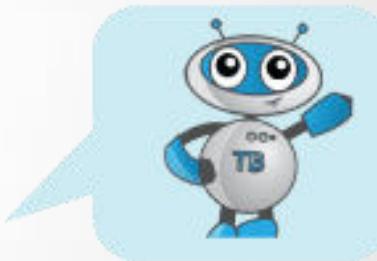
Appium



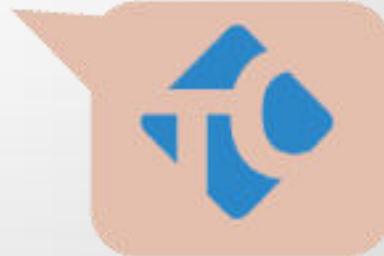
Kobiton



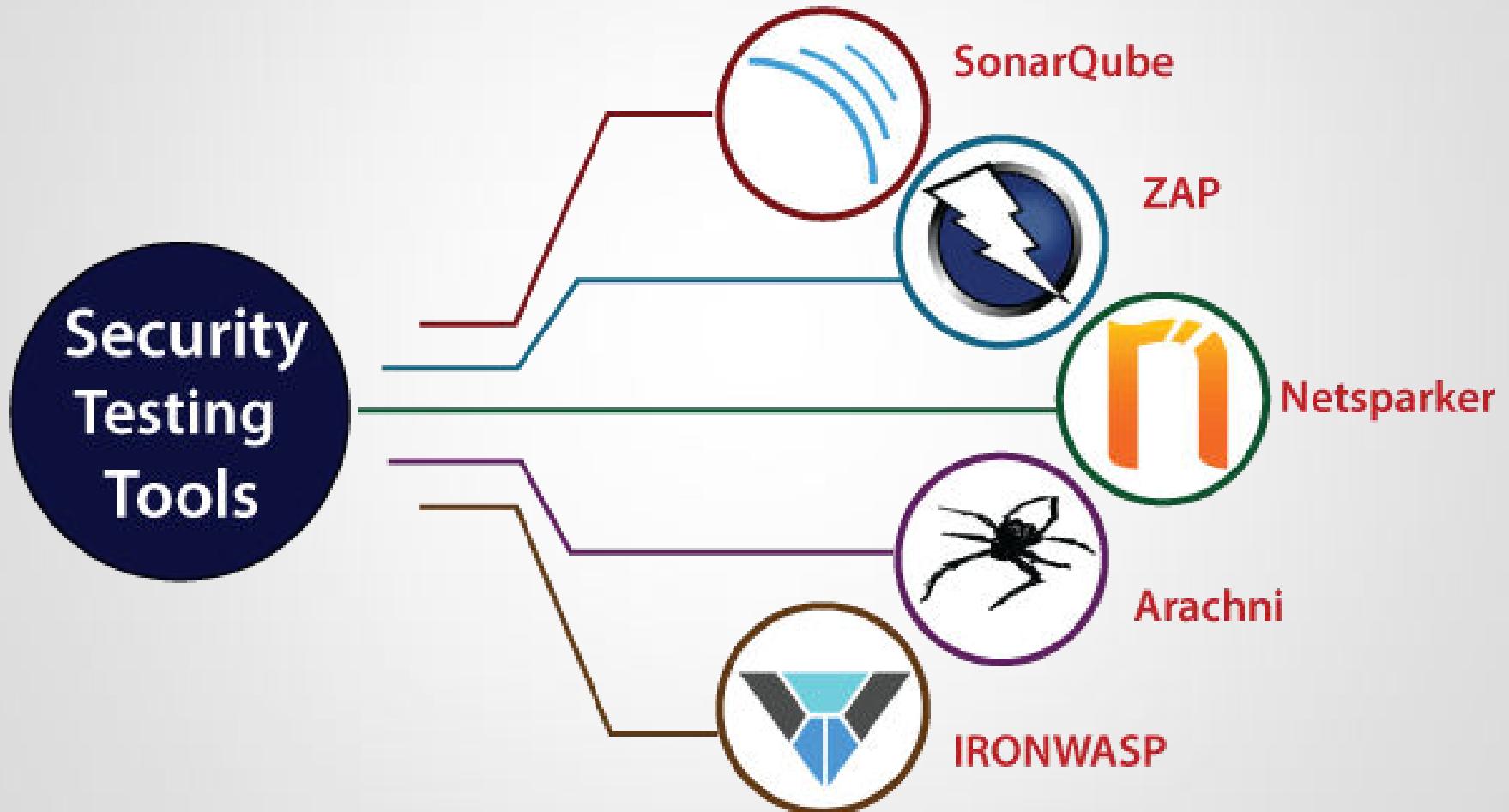
TestingBot



TestComplete



Testing Tools



Testing Tools



Testing Documentation

Before Testing

Testing starts with test cases generation. Following documents are needed for reference –

- **SRS document** - Functional Requirements document
- **Test Policy document** - This describes how far testing should take place before releasing the product.
- **Test Strategy document** - This mentions detail aspects of test team, responsibility matrix and rights/responsibility of test manager and test engineer.
- **Traceability Matrix document** - This is SDLC document, which is related to requirement gathering process. As new requirements come, they are added to this matrix. These matrices help testers know the source of requirement. They can be traced forward and backward.

Testing Documentation

While Being Tested

The following documents may be required while testing is started and is being done:

- **Test Case document** - This document contains list of tests required to be conducted. It includes Unit test plan, Integration test plan, System test plan and Acceptance test plan.
- **Test description** - This document is a detailed description of all test cases and procedures to execute them.
- **Test case report** - This document contains test case report as a result of the test.
- **Test logs** - This document contains test logs for every test case report.

Testing Documentation

After Testing

The following documents may be generated after testing :

- **Test summary** - This test summary is collective analysis of all test reports and logs. It summarizes and concludes if the software is ready to be launched. The software is released under version control system if it is ready to launch.

Software Quality

Software Quality

Software Quality

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



Software Quality

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely **made to work in various operating system environments**, in multiple machines, with other software products, etc.

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Testing vs. QA, QC and Audit

Software quality assurance (QA) -

These are software development process monitoring means, by which it is assured that **all the measures are taken as per the standards of organization**. This monitoring is done to make sure that **proper software development methods were followed**.

The responsibility of quality assurance is not of any specific team, but it is a responsibility of each member of the development team.

- Quality assurance **prevents defects**.
- Quality assurance is **process oriented**.
- Quality assurance is proactive in a process and preventive in nature.
- Quality assurance is a managerial tool.
- Each developer is responsible for quality assurance.

Testing vs. QA, QC and Audit

Software quality control (QC) - This is a system to maintain the quality of software product. It may include **functional and non-functional** aspects of software product, which enhance the goodwill of the organization. This system makes sure that the customer is receiving quality product for their requirement and the product certified as '**fit for use**'.

The responsibility of quality control is of a specific team which is known as a testing team that tests the defects of software by validation and corrective tools.

- Quality Control provides **identification of defects**.
- Quality Control is **product** oriented.
- Quality Control is a corrective tool.
- Testing team is responsible for Quality control.
- Quality Control is a reactive process.

Testing vs. QA, QC and Audit

Points	Quality Assurance (QA)	Quality Control (QC)
Definition	QA is a group of activities which ensures that the quality of processes which is used during the development of the software always be maintained.	QC is a group of activities to detect the defects in the developed software.
Focus	The focus of QA is to prevent defects in the developing software by paying attention to processes.	The focus of QC is to identify defects in the developed software by paying attention to testing processes.
How	Establishment of the high-quality management system and periodic audits for conformance of the operations of the developing software.	Detecting and eliminating the quality problem elements by using testing techniques and tools in the developed software.
What	QA ensures prevention of quality problem elements by using systematic activities including documentation.	QC ensures identification and elimination of defects by using processes and techniques to achieve and maintain high quality of the software.
Orientat ion	QA is process oriented .	QC is product oriented .
Type of process	QA is a proactive process. It concerns to improve development so; defects do not arise in the testing period.	QC is a reactive process because it concerns to identify defects after the development of product and before its release.
Responsibility	Each and every member of the development team is responsible for QA	Only the specific testing team is responsible for QC
Exampler	Verification is the example of QA	Validation is the example of QC

Testing vs. QA, QC and Audit

Software audit - This is a **review of procedure** used by the organization to develop the software. A team of auditors, **independent of development team examines the software process, procedure, requirements and other aspects of SDLC**. The purpose of software audit is to check that software and its development process, both conform standards, rules and regulations.

ISO 9000 Certification

ISO 9000 Certification

ISO (International Standards Organization) is a group or consortium of **63 countries** established to plan and fosters standardization. ISO declared its **9000 series of standards in 1987**. It serves as a reference for the contract between independent parties. The ISO 9000 standard determines the **guidelines for maintaining a quality system**. The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc. ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself.

ISO 9000 Certification

Types of ISO 9000 Quality Standards

ISO 9000 is a series of three standards:



ISO 9000 Certification

How to get ISO 9000 Certification?



How to get ISO 9000 Certification?

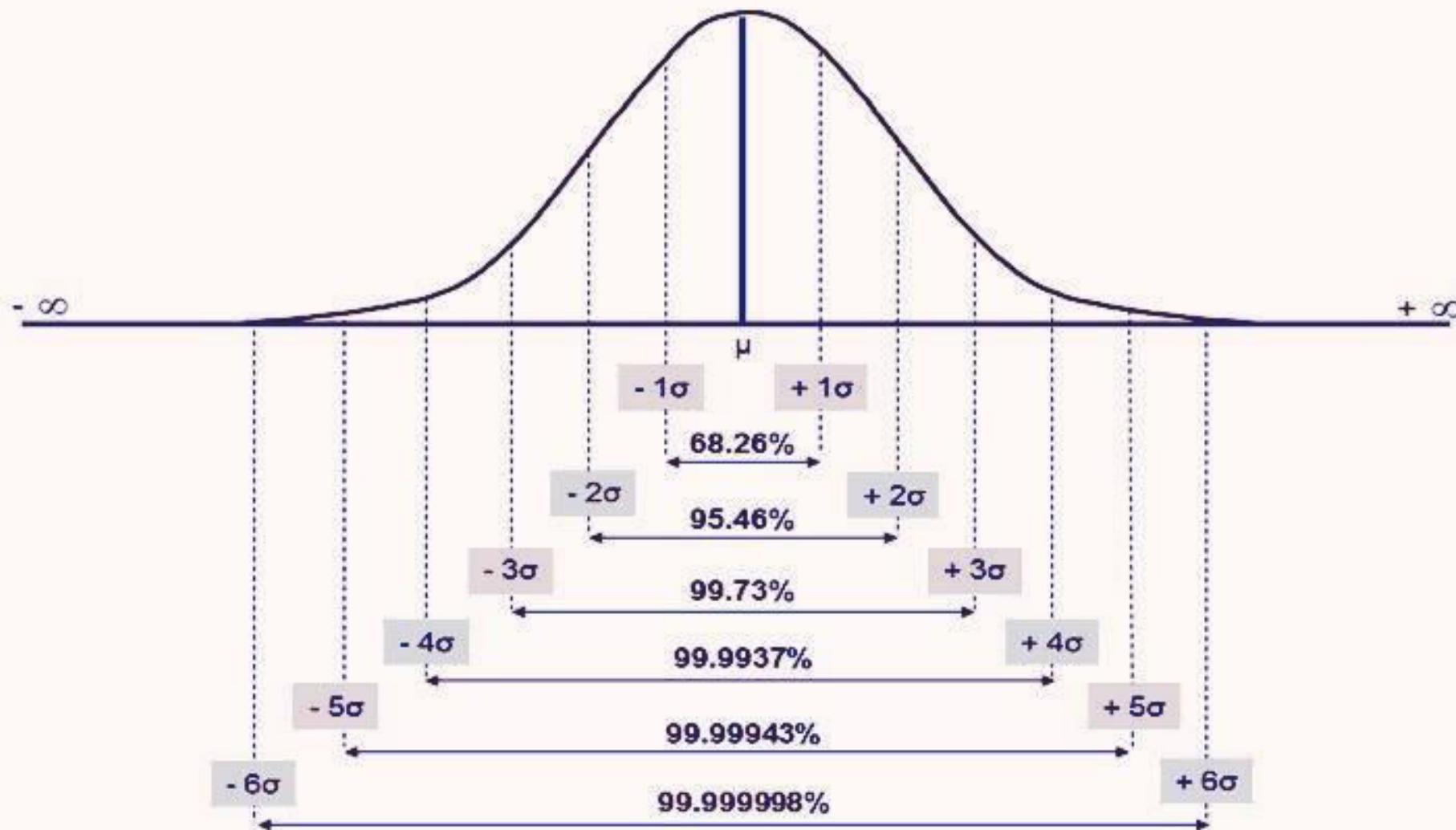
- ❖ **Application:** Once an organization decided to go for ISO certification, it applies to the registrar for registration.
- ❖ **Pre-Assessment:** During this stage, the registrar makes a rough assessment of the organization.
- ❖ **Document review and Adequacy of Audit:** During this stage, the registrar reviews the document submitted by the organization and suggest an improvement.
- ❖ **Compliance Audit:** During this stage, the registrar checks whether the organization has compiled the suggestion made by it during the review or not.
- ❖ **Registration:** The Registrar awards the ISO certification after the successful completion of all the phases.
- ❖ **Continued Inspection:** The registrar continued to monitor the organization time by time.

Six Sigma

Six Sigma

Six Sigma is the process of **improving the quality of the output by identifying and eliminating the cause of defects and reduce variability in manufacturing and business processes**. The maturity of a manufacturing process can be defined by a **sigma rating** indicating its percentage of defect-free products it creates. A six sigma method is one in which **99.99966%** of all the opportunities to produce some features of a component are statistically expected to be free of defects (**3.4 defective features per million opportunities**).

Six Sigma

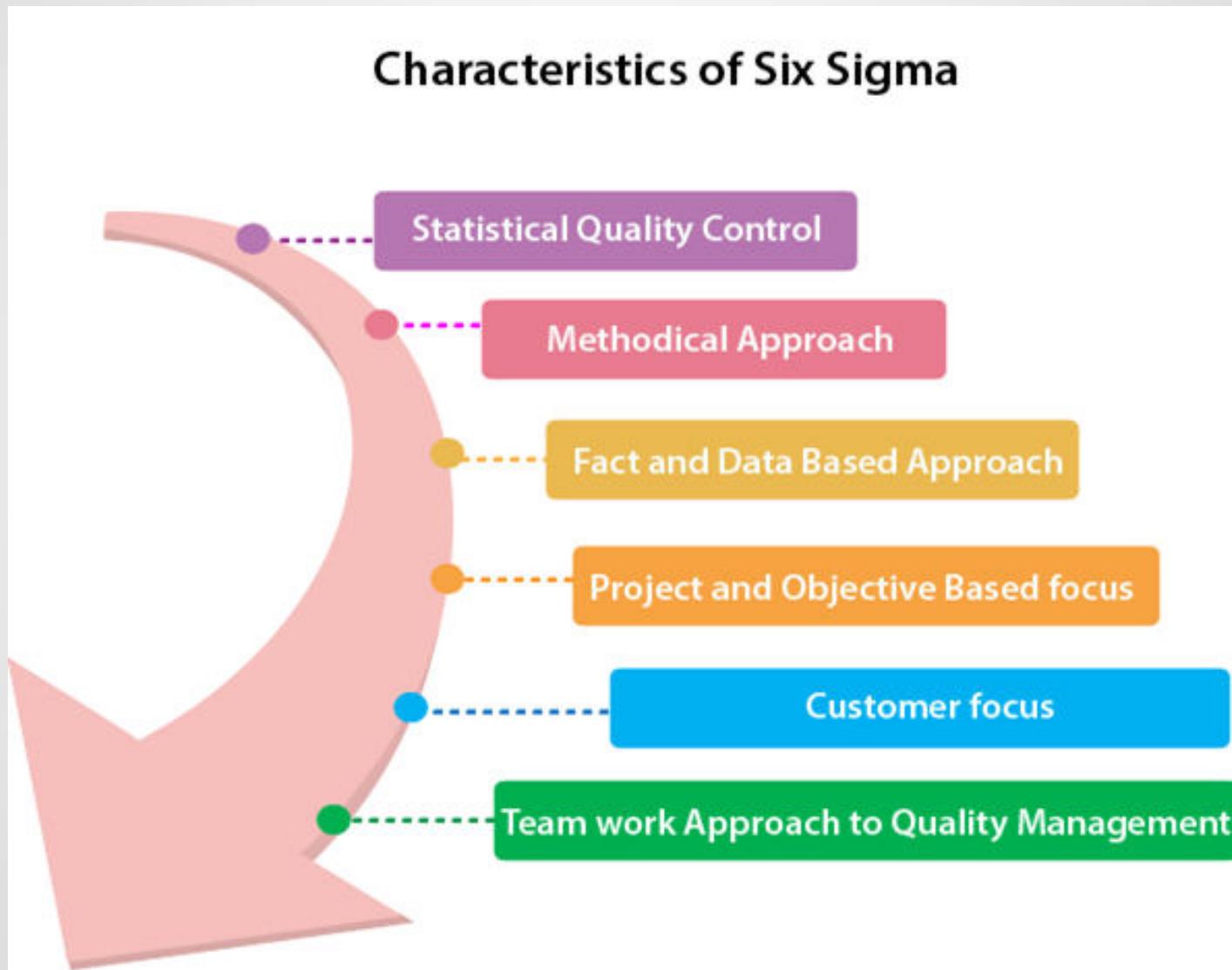


Origin of Six Sigma

- Six Sigma originated at Motorola in the early 1980s, in response to achieving 10X reduction in product-failure levels in 5 years.
- Engineer **Bill Smith** invented Six Sigma, but died of a heart attack in the Motorola cafeteria in 1993, never knowing the scope of the craze and controversy he had touched off.
- Six Sigma is based on various quality management theories (e.g. Deming's 14 point for management, Juran's 10 steps on achieving quality).

Six Sigma

Characteristics of Six Sigma



Six Sigma

Characteristics of Six Sigma

- ❖ **Statistical Quality Control:** Six Sigma is derived from the Greek Letter σ (Sigma) from the Greek alphabet, which is used to denote Standard Deviation in statistics. Standard Deviation is used to measure variance, which is an essential tool for measuring non-conformance as far as the quality of output is concerned.
- ❖ **Methodical Approach:** The Six Sigma is not a merely quality improvement strategy in theory, as it features a well defined systematic approach of application in DMAIC and DMADV which can be used to improve the quality of production. DMAIC is an acronym for Design-Measure- Analyze-Improve-Control. The alternative method DMADV stands for Design-Measure- Analyze-Design-Verify.
- ❖ **Fact and Data-Based Approach:** The statistical and methodical aspect of Six Sigma shows the scientific basis of the technique. This accentuates essential elements of the Six Sigma that is a fact and data-based.

Six Sigma

Characteristics of Six Sigma

- ❖ **Project and Objective-Based Focus:** The Six Sigma process is implemented for an organization's project tailored to its specification and requirements. The process is flexed to suit the requirements and conditions in which the projects are operating to get the best results.
- ❖ **Customer Focus:** The customer focus is fundamental to the Six Sigma approach. The quality improvement and control standards are based on specific customer requirements.
- ❖ **Teamwork Approach to Quality Management:** The Six Sigma process requires organizations to get organized when it comes to controlling and improving quality. Six Sigma involving a lot of training depending on the role of an individual in the Quality Management team.

Six Sigma

Benefits of Six Sigma

Six Sigma offers six major benefits that attract companies –

- ❖ Generates sustained success
- ❖ Sets a performance goal for everyone
- ❖ Enhances value to customers
- ❖ Accelerates the rate of improvement
- ❖ Promotes learning and cross-pollination
- ❖ Executes strategic change

Six Sigma

Six Sigma Methodologies

Six Sigma projects follow two project methodologies:

1. **DMAIC:** is used to enhance an existing business process
2. **DMADV:** is used to create new product designs or process designs



Software Maintenance

Outlines

- ✧ **Software Maintenance**
- ✧ **Need for Maintenance**
- ✧ **Types of Software Maintenance**
- ✧ **Causes of Software Maintenance Problems**
- ✧ **Software Maintenance Cost Factors**
- ✧ **Software Re-engineering**

Software Maintenance

Software Maintenance is the process of **modifying** a software product after it has been delivered to the customer.

The main purpose of software maintenance is to **modify and update software application** after delivery to correct faults and to improve performance.



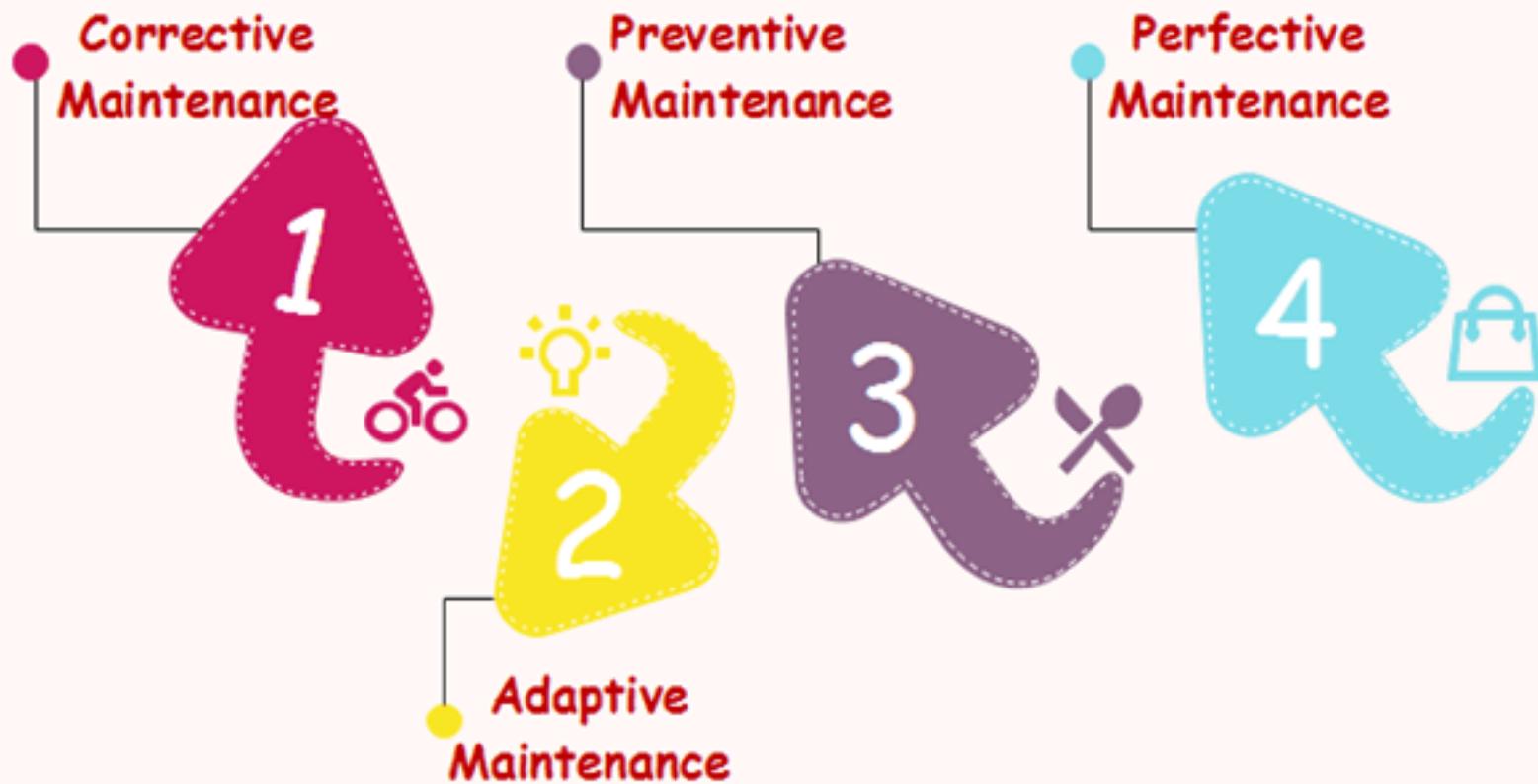
Need for Maintenance

Software Maintenance is required to ensure that the system continues to satisfy user requirements:-

- Correct errors
- Change in user requirement with time
- Changing hardware/software requirements
- To improve system efficiency
- To optimize the code to run faster
- To modify the components
- To reduce any unwanted side effects.

Types of Software Maintenance

Software Maintenance is classified in the following categories:



Types of Software Maintenance

Maintenance can be divided into the following:

1. Corrective maintenance:

Corrective maintenance of a software product may be essential either to **rectify some bugs observed** while the system is in use, or to enhance the performance of the system.

2. Adaptive maintenance:

This includes modifications and updates when the customers need the product to run on **new platforms, on new operating systems**, or when they need the product to interface with new hardware and software.

3. Perfective maintenance:

A software product needs maintenance to **support the new features** that the users want or to change different types of functionalities of the system according to the customer demands.

4. Preventive maintenance:

This type of maintenance includes modifications and updates **to prevent future problems** of the software. It goals to attend problems, **which are not significant at this moment but may cause serious issues in future.**

Causes of Software Maintenance Problems

Following are the major causes of software maintenance problems:

01

Lack of
Traceability

02

Lack of
code
comments

03

Obsolete
Legacy
Systems

Causes of Software Maintenance Problems

Lack of Traceability

- Codes are rarely traceable to the requirements and design specifications.
- It makes it **very difficult for a programmer to detect and correct a critical defect affecting customer operations.**
- Like a detective, the programmer pores over the program looking for clues.
- Life Cycle documents are not always produced even as part of a development project.

Lack of Code Comments

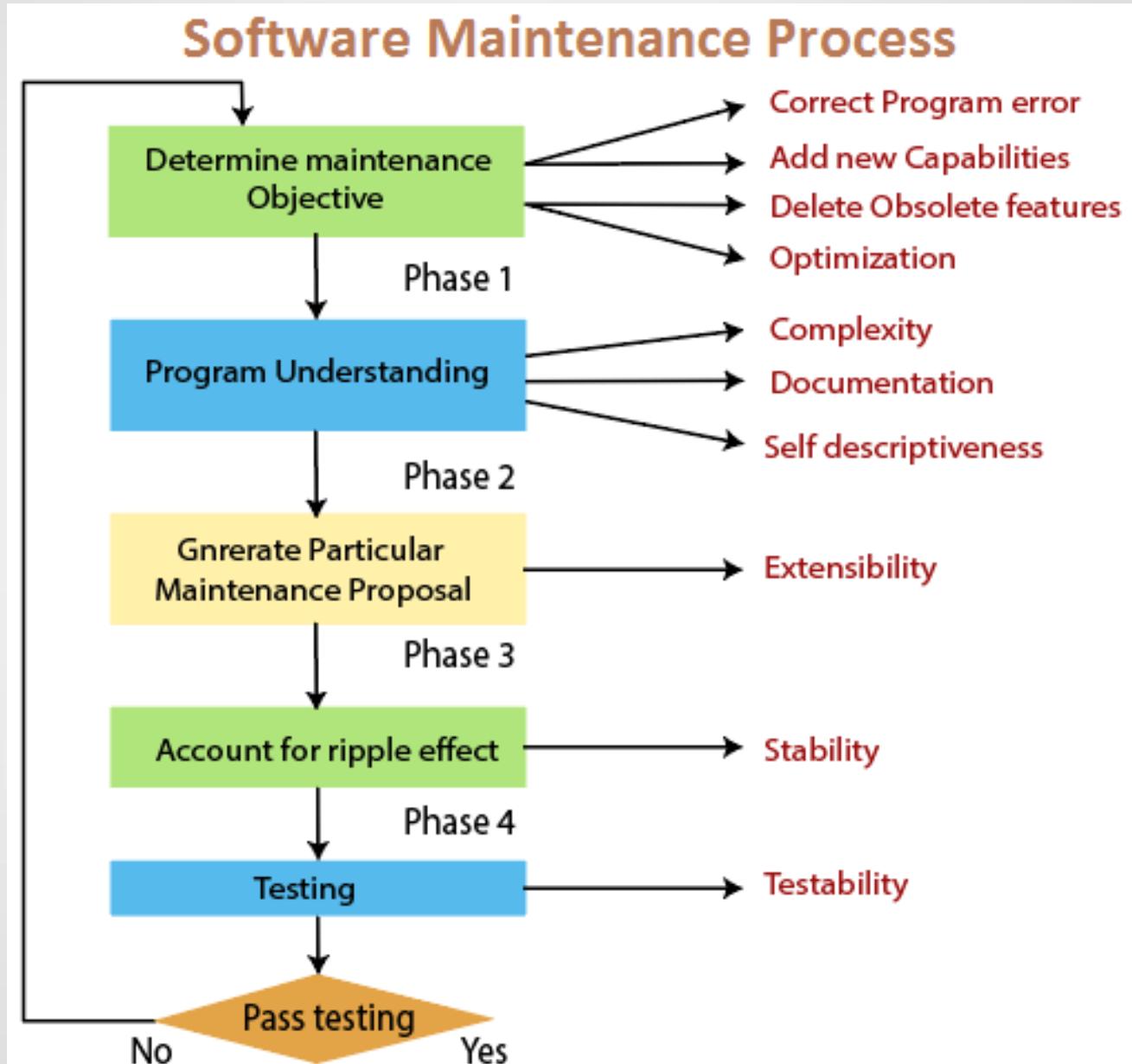
- Most of the software system codes lack adequate comments. Lesser comments may not be helpful in certain situations.

Causes of Software Maintenance Problems

Obsolete Legacy Systems

- In most of the countries worldwide, the legacy system that provides the backbone of the nation's critical industries, e.g., telecommunications, medical, transportation utility services, were not designed with maintenance in mind.
- They were not expected to last for a quarter of a century or more.
- As a consequence, the code supporting these systems is devoid of traceability to the requirements, compliance to design and programming standards and often includes dead, extra and uncommented code, which all make the maintenance task next to the impossible.

Causes of Software Maintenance Problems



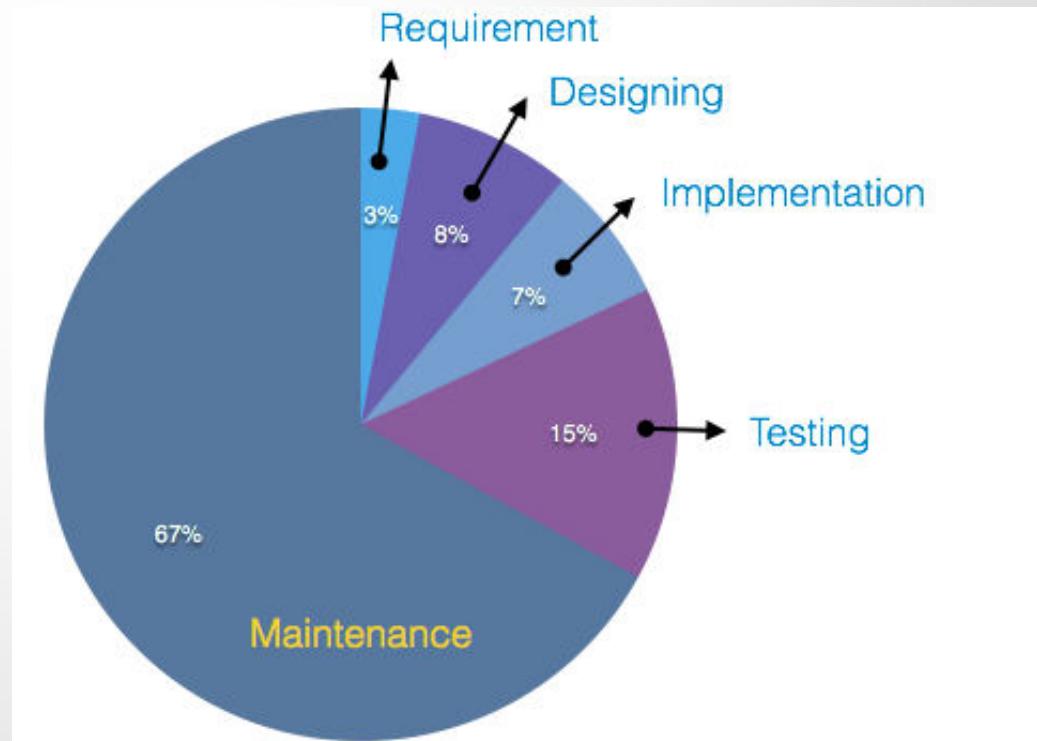
Software Maintenance Cost Factors

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

There are two types of cost factors involved in software maintenance.

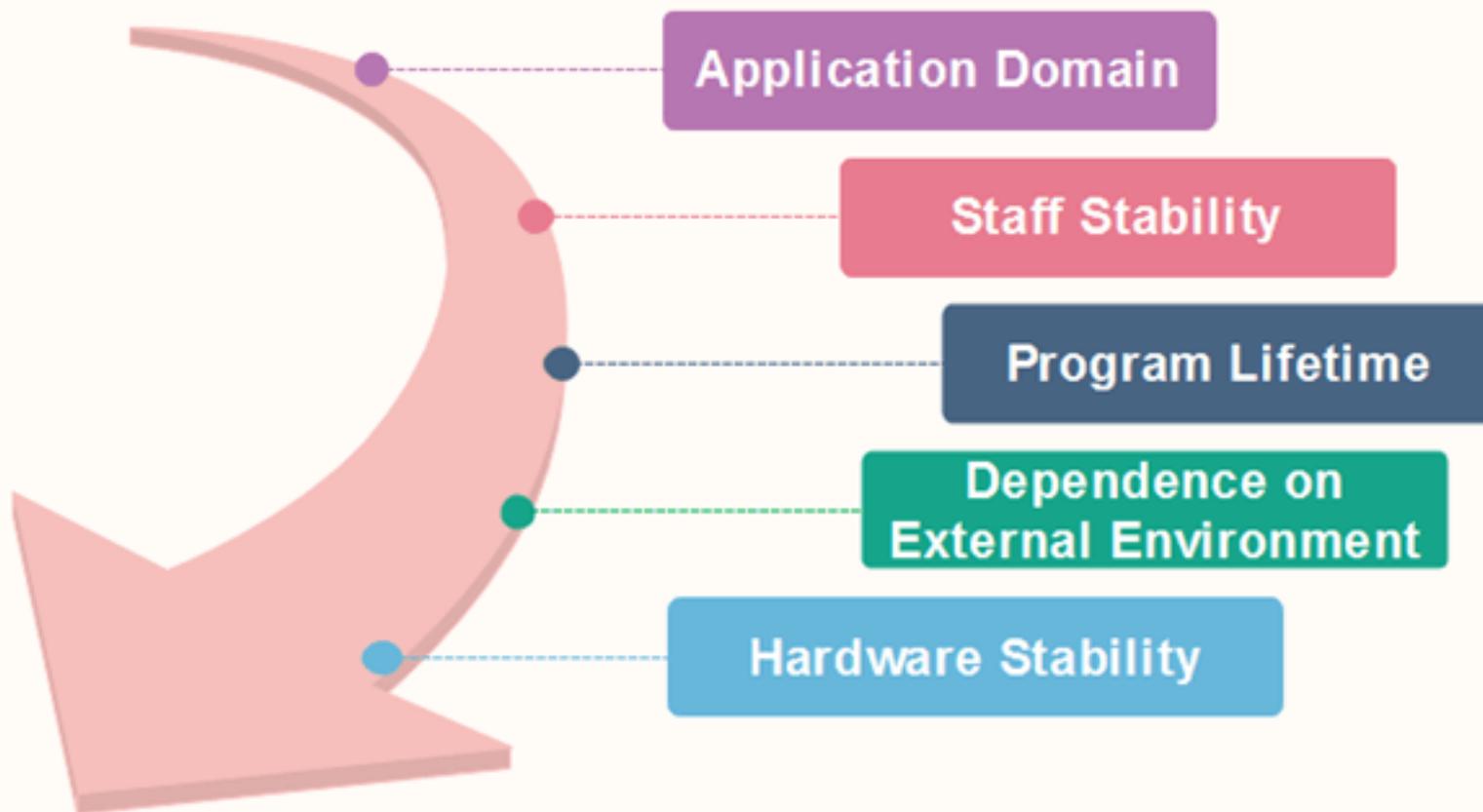
These are

- Non-Technical Factors
- Technical Factors



Software Maintenance Cost Factors

The non-technical factors include



Software Maintenance Cost Factors

Technical Factors

Module
Independence

01

Programming
Language

02

Programming
Style

03

Program Validation
& Testing

04

Documentation

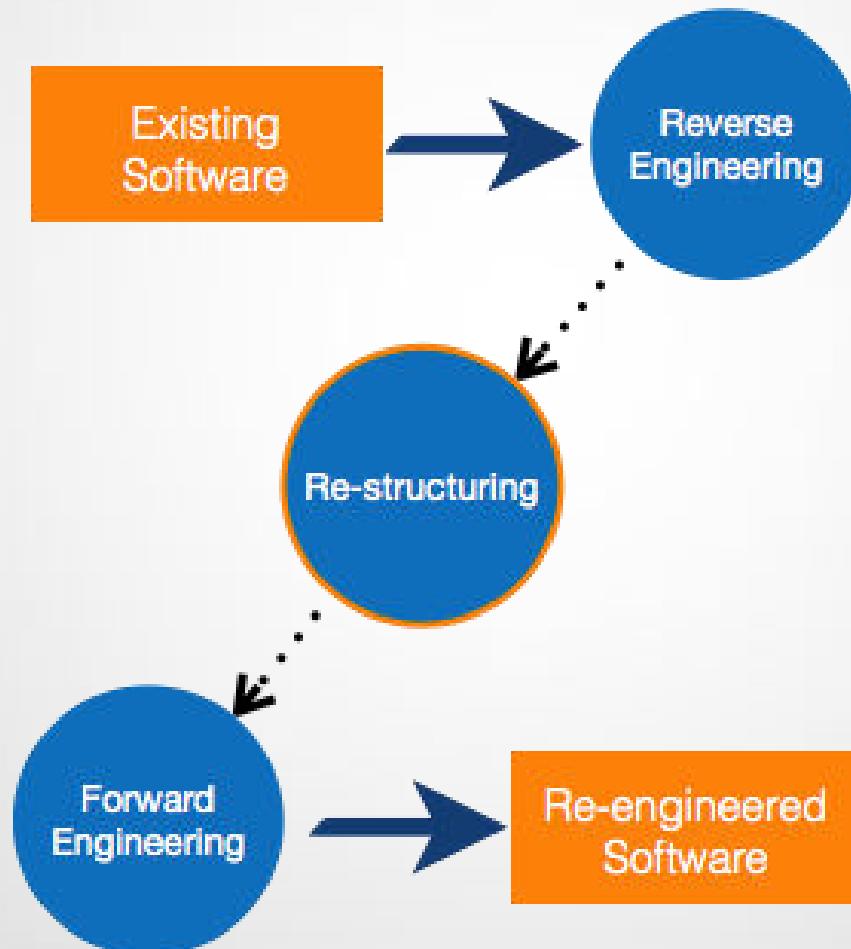
05

Configuration
Management

06

Software Re-engineering

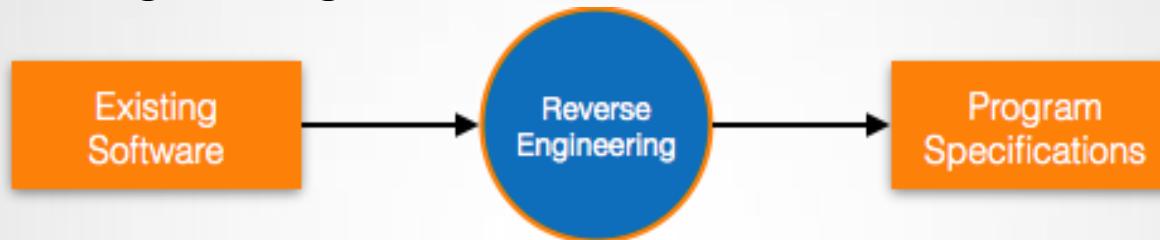
When we need to update the software to keep it to the current market, without impacting its functionality, it is called **software re-engineering**. It is a thorough process where the design of software is changed and programs are re-written.



Re-Engineering Process

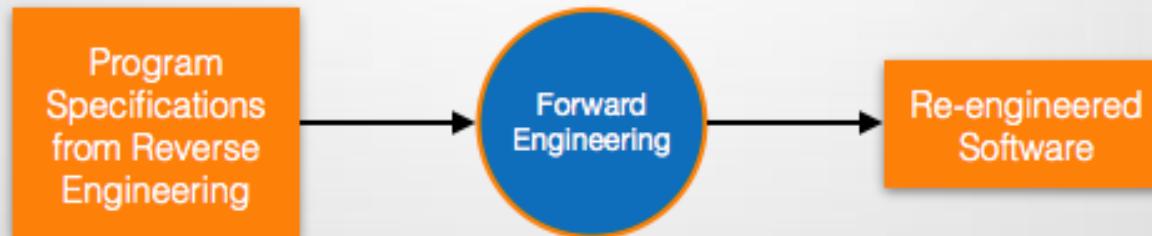
Reverse Engineering

Reverse Engineering is processes of extracting knowledge or design information from anything man-made and reproducing it based on extracted information. It is also called back Engineering.



Forward Engineering

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.



Re-Engineering Process

Why Reverse Engineering?

- Providing proper system documentation.
- Recovery of lost information.
- Assisting with maintenance.
- Facility of software reuse.
- Discovering unexpected flaws or faults

Used of Software Reverse Engineering –

- Software Reverse Engineering is used in software design, reverse engineering enables the developer or programmer to add new features to the existing software with or without knowing the source code.
- Reverse engineering is also useful in software testing, it helps the testers to study the virus and other malware code .



CSE- 321

Software Engineering

Security



Software Security



Software Security

What does Software Security mean?

In the general sense, security is “the state of **being free from danger or threat**”. The security of software systems in particular is a vast topic.

Software security is the application of techniques that assess, mitigate, and protect software systems from vulnerabilities.

These techniques ensure that software continues to function and are safe from attacks. Developing secure software involves considering security at every stage of the life cycle.

Software security isn't the same thing as **security software**.

Software Security

What does Software Security mean?

Security testing takes the following six measures to provide a secured environment



1. **Confidentiality** – It protects against disclosure of information to unintended recipients.
2. **Integrity** – It allows transferring accurate and correct desired information from senders to intended receivers.
3. **Availability** – It ensures readiness of the information on requirement.
4. **Non-repudiation** – It ensures there is no denial from the sender or the receiver for having sent or received the message.
5. **Authentication** – It verifies and confirms the identity of the user.
6. **Authorization** – It specifies access rights to the users and resources.

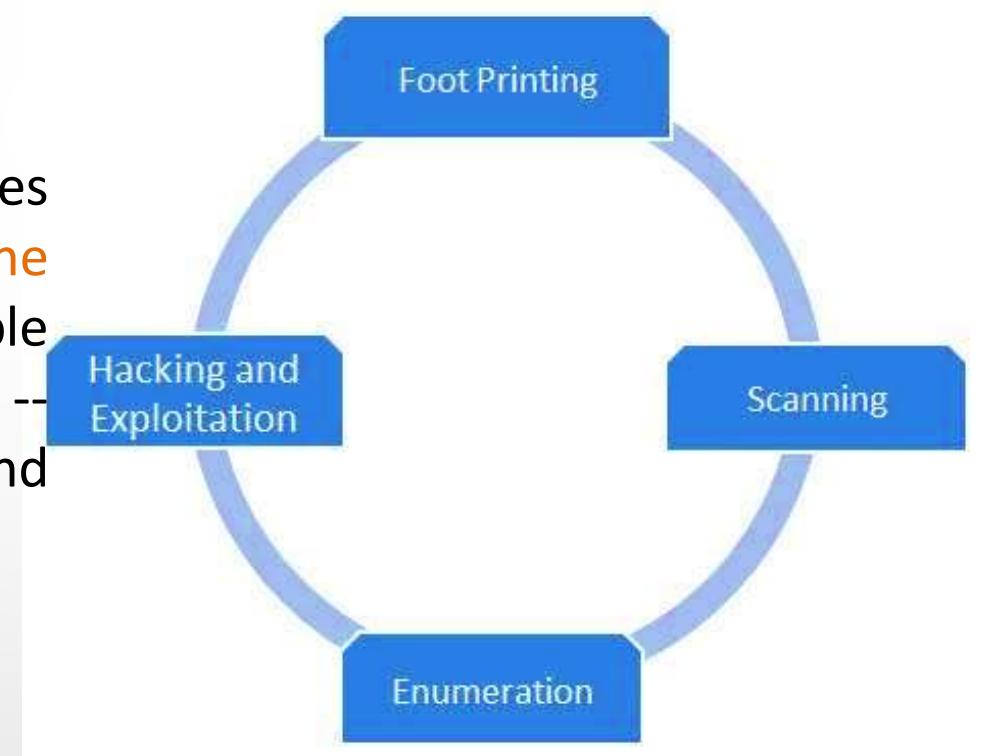
Software Security

Penetration testing, also called **pen testing** or **ethical hacking**, is the practice of testing a computer system, network or web application to **find security vulnerabilities that an attacker could exploit**.

The main objective of penetration testing is **to identify security weaknesses**.

Penetration testing can be automated with software applications or performed manually.

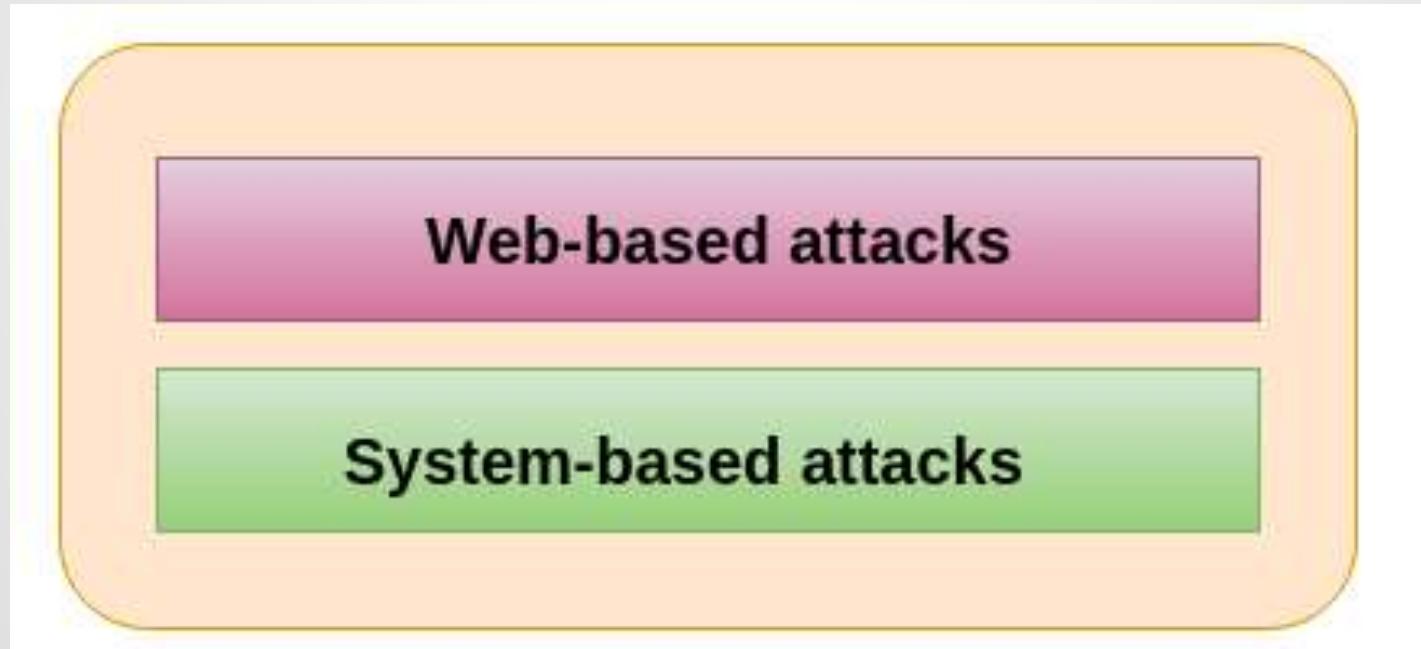
Either way, the process involves **gathering information about the target** before identifying possible entry points, attempting to break in -- either virtually or for real -- and reporting back the findings.



Malicious software (malware)

Malicious software (malware) is any software that gives partial to full control of the system to the attacker/malware creator.

Types of Attacks



Web-based attacks

Web-based attacks

These are the attacks which occur on a website or web applications. Some of the important web-based attacks are as follows-

Injection attacks

It is the attack in which **some data will be injected into a web application** to manipulate the application and fetch the required information.

Example- SQL Injection, code Injection, log Injection, XML Injection etc.

DNS Spoofing

DNS Spoofing is a type of computer security hacking. Whereby a **data** is introduced into a **DNS resolver's cache causing the name server to return an incorrect IP address**, **diverting traffic to the attacker's computer or any other computer**. The DNS spoofing attacks can go on for a long period of time without being detected and can cause serious security issues.

Session Hijacking

It is a security attack on a user session over a protected network. Web applications create cookies to store the state and user sessions. **By stealing the cookies, an attacker can have access to all of the user data.**

Phishing

Phishing is a type of attack which attempts to steal sensitive information like user login credentials and credit card number. It occurs when an attacker is masquerading as a trustworthy entity in electronic communication.

Brute force

It is a type of attack which uses a trial and error method. This attack generates a large number of guesses and validates them to obtain actual data like user password and personal identification number. This attack may be used by criminals to crack encrypted data, or by security analysts to test an organization's network security.

Denial of Service (DOS)

It is an attack which meant to make a server or network resource unavailable to the users. It accomplishes this by flooding the target with traffic or sending it information that triggers a crash. It uses the single system and single internet connection to attack a server. It can be classified into the following-

Volume-based attacks- Its goal is to saturate the bandwidth of the attacked site, and is measured in bit per second.

Protocol attacks- It consumes actual server resources, and is measured in a packet.

Application layer attacks- Its goal is to crash the web server and is measured in request per second.

Web-based attacks

Dictionary attacks

This type of attack stored the list of a commonly used password and validated them to get original password.

URL Interpretation

It is a type of attack where we can change the certain parts of a URL, and one can make a web server to deliver web pages for which he is not authorized to browse.

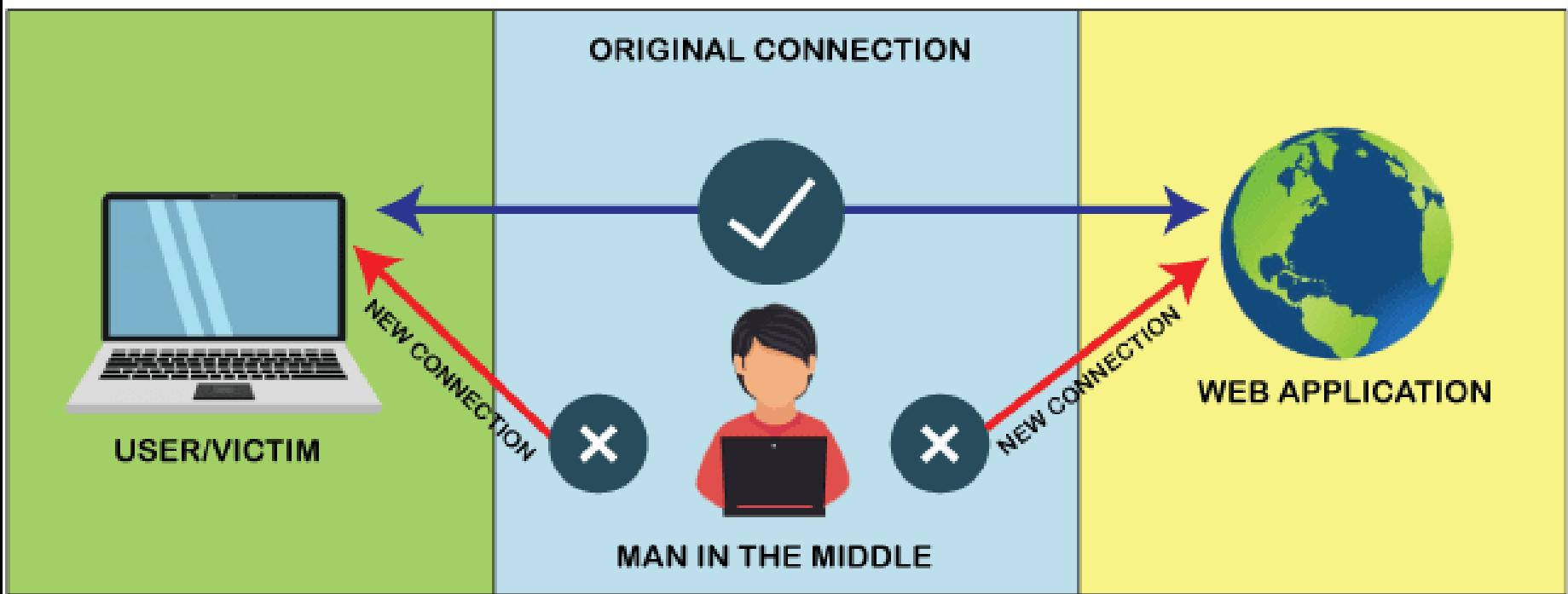
File Inclusion attacks

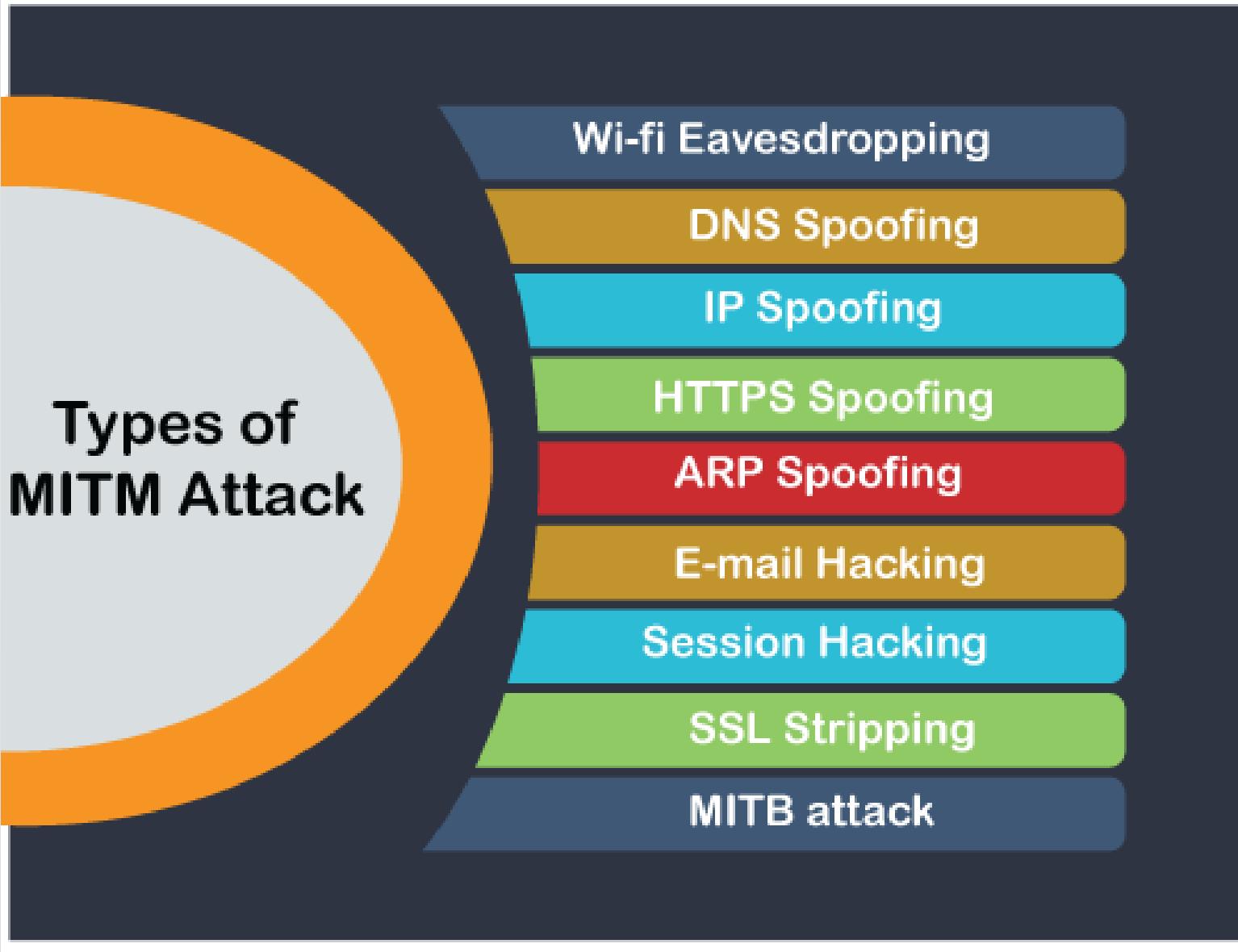
It is a type of attack that allows an attacker to access unauthorized or essential files which is available on the web server or to execute malicious files on the web server by making use of the include functionality.

Man in the middle attacks(MITM)

It is a type of attack that allows an attacker to intercepts the connection between client and server and acts as a bridge between them. Due to this, an attacker will be able to read, insert and modify the data in the intercepted connection.

HOW MAN IN THE MIDDLE ATTACKS WORK





System-based attacks

System-based attacks

These are the attacks which are intended to compromise a computer or a computer network. Some of the important system-based attacks are as follows-

Virus

It is a type of **malicious software** program that spread throughout the computer files **without the knowledge of a user**. It is a **self-replicating** malicious computer program that replicates by inserting copies of itself into other computer programs when executed. It can also execute instructions that cause harm to the system.



Worm

It is a type of malware whose primary function is to **replicate itself** to spread to **uninfected computers**. It works same as the computer virus. Worms often originate from email attachments that appear to be from trusted senders.

Web-based attacks

Adware – Adware, also known as **freeware or pitchware**, is a free computer software that **contains commercial advertisements** of games, desktop toolbars, and utilities. It is a web-based application and it **collects web browser data to target advertisements, especially pop-ups**.

Spyware – Spyware is infiltration software that **anonymously monitors users** which enables a hacker to obtain sensitive information from the user's computer. Spyware exploits users and application vulnerabilities that is quite often attached to free online software downloads or to links that are clicked by users.

Ransomware - is a type of malware from **cryptovirology** that threatens to publish the victim's data or **perpetually block access to it unless a ransom is paid**. While some simple ransomware may lock the system so that it is not difficult for a knowledgeable person to reverse, more advanced malware uses a technique called cryptoviral extortion. It encrypts the victim's files, making them inaccessible, and demands a ransom payment to decrypt them

Web-based attacks

Trojan horse

It is a malicious program that occurs unexpected changes to computer setting and unusual activity, even when the computer should be idle. It misleads the user of its true intent. It appears to be a normal application but when opened/executed some malicious code will run in the background.



Backdoors

It is a method that bypasses the normal authentication process. A developer may create a backdoor so that an application or operating system can be accessed for troubleshooting or other purposes.



Web-based attacks

Rootkit – A rootkit is a software used by a hacker to gain admin level access to a computer/network which is installed through a stolen password or by exploiting a system vulnerability without the victim's knowledge.



Bots

A bot (short for "robot") is an automated process that interacts with other network services. Some bots program run automatically, while others only execute commands when they receive specific input. Common examples of bots program are the crawler, chatroom bots, and malicious bots.



KeyLoggers:

Traditionally, Keyloggers are software that monitor user activity such as keys typed using keyboard.

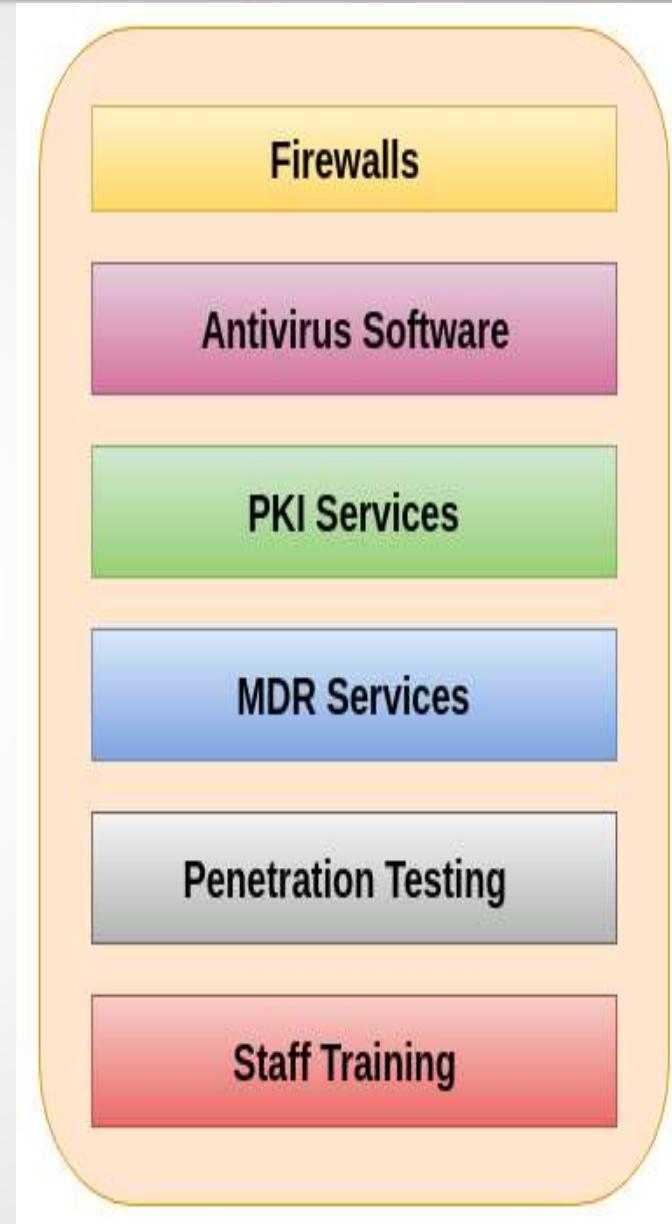
Modern keyloggers can,

- Record keystrokes on keyboard
- Record mouse movement and clicks
- Record menus that are invoked
- Take screenshots of the desktop at predefined intervals



Security Technologies

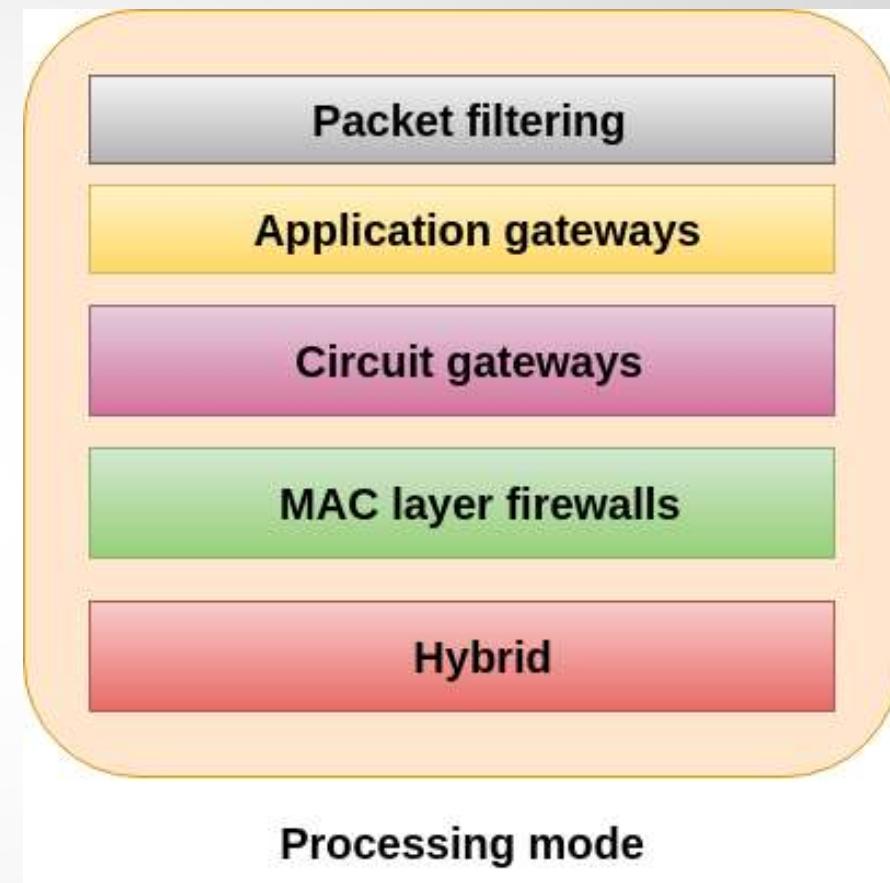
1. Firewalls
2. Antivirus Software
3. PKI Services (Public Key Infrastructure)
4. Managed Detection and Response Service
(MDR)
5. Penetration Testing
6. Staff Training



Security Technologies

Firewall

Firewall is a computer **network security system** designed to prevent unauthorized access to or from a private network. It can be implemented as hardware, software, or a combination of both. Firewalls are used to prevent unauthorized Internet users from accessing private networks connected to the Internet. All messages are entering or leaving the intranet pass through the firewall. The firewall examines each message and blocks those that do not meet the specified security criteria.



Security Technologies

Intrusion Detection System (IDS)

An IDS is a security system which monitors the computer systems and network traffic.

It analyses that traffic for possible hostile attacks originating from the outsider and also for system misuse or attacks originating from the insider.

A firewall does a job of filtering the incoming traffic from the internet, the IDS in a similar way complements the firewall security. Like, the firewall protects an organization sensitive data from malicious attacks over the Internet, the Intrusion detection system alerts the system administrator in the case when someone tries to break in the firewall security and tries to have access on any network in the trusted side.

Security Technologies

Access Control

Access control is a process of selecting restrictive access to a system. It is a concept in security to minimize the risk of unauthorized access to the business or organization.

In this, users are granted access permission and certain privileges to a system and resources. Here, users must provide the credential to be granted access to a system. These credentials come in many forms such as password, keycard, the biometric reading, etc. Access control ensures security technology and access control policies to protect confidential information like customer data.

The access control can be categorized into two types-

- Physical access control
- Logical access control

Security Technologies

Encoding and Decoding

Encoding is the process of putting a sequence of characters such as letters, numbers and other special characters into a specialized format for efficient transmission.

Decoding is the process of converting an encoded format back into the original sequence of characters. It is completely different from Encryption which we usually misinterpret.

Encoding should NOT be used for transporting sensitive information.

Cryptography is useful tool for securing data, communications, and code globs, but it's no silver bullet.

Security Technologies

Security orchestration, automation and response (SOAR) software solutions combine the functionality of security tools and add intelligent automation functionality to improve a security operations team's ability to tackle threats in real time with minimal manual labour.

User and entity behaviour analytics (UEBA) software and zero trust networking solutions are designed to detect threats in real time. These tools all use behaviour-based analytics to identify strange behaviours and misuse within a network.

Digital Signature

A digital signature is a **mathematical technique** which validates the **authenticity and integrity** of a message, software or **digital documents**. It allows us to verify the author name, date and time of signatures, and authenticate the message contents. The digital signature offers far more **inherent security** and intended to solve the problem of tampering and impersonation (Intentionally copy another person's characteristics) in digital communications.

Algorithms in Digital Signature

A digital signature consists of three algorithms:

1. Key generation algorithm

The key generation algorithm selects private key randomly from a set of possible private keys. This algorithm provides the private key and its corresponding public key.

2. Signing algorithm

A signing algorithm produces a signature for the document.

3. Signature verifying algorithm

A signature verifying algorithm either accepts or rejects the document's authenticity.

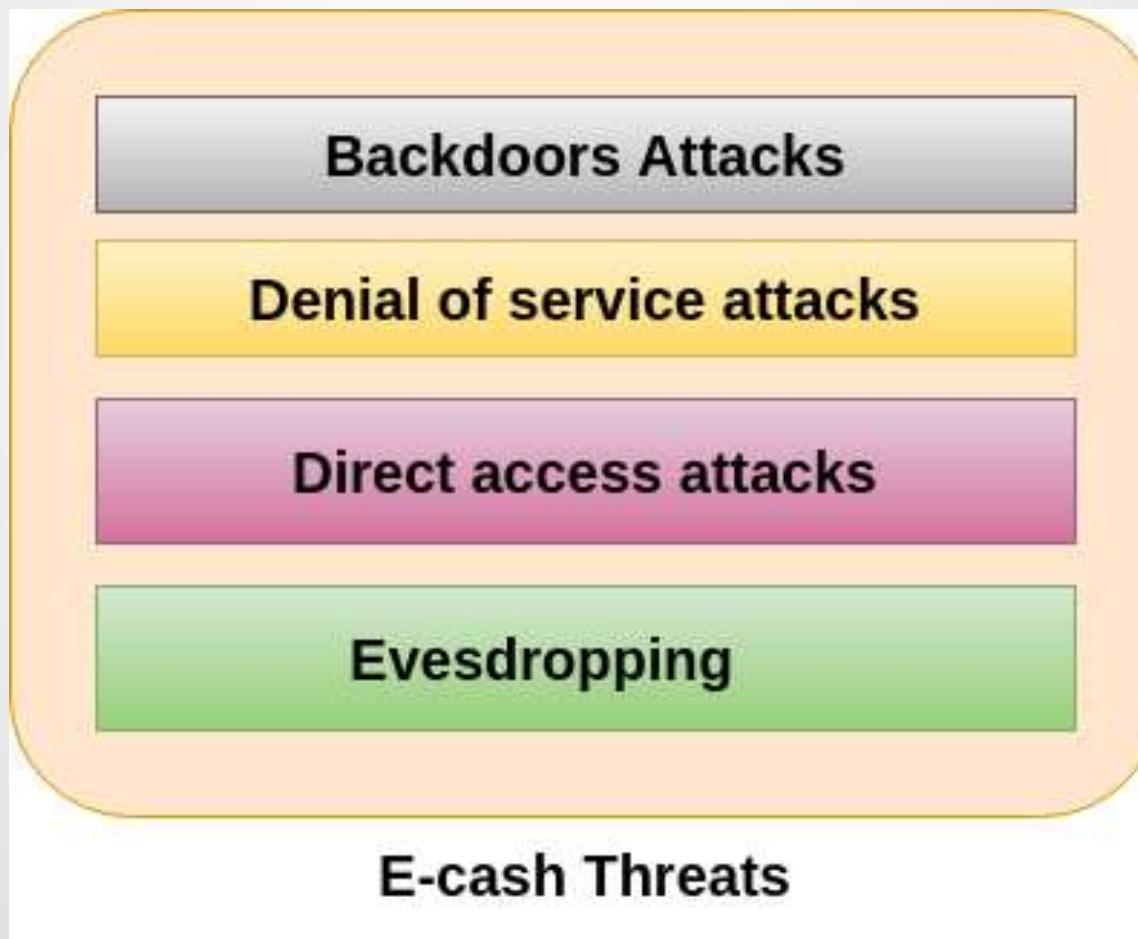
Threat to E-Commerce

Major Threats to E-Commerce Industry



Threat to E-Commerce

In e-cash, we stored financial information on the electronic device or on the internet which is vulnerable to the hackers. Some of the major threats related to e-cash system are-



Threat to E-Commerce

Backdoors Attacks

It is a type of attacks which gives an attacker to unauthorized access to a system by bypasses the normal authentication mechanisms. It works in the background and hides itself from the user that makes it difficult to detect and remove.

Denial of service attacks

A denial-of-service attack (DoS attack) is a security attack in which the attacker takes action that prevents the legitimate (correct) users from accessing the electronic devices. It makes a network resource unavailable to its intended users by temporarily disrupting services of a host connected to the Internet.

Direct Access Attacks

Direct access attack is an attack in which an intruder gains physical access to the computer to perform an unauthorized activity and installing various types of software to compromise security. These types of software loaded with worms and download a huge amount of sensitive data from the target victims.

Threat to E-Commerce

Eavesdropping

This is an **unauthorized way of listening to private communication over the network**. It does not interfere with the normal operations of the targeting system so that the sender and the recipient of the messages are not aware that their conversation is tracking.

Credit/Debit card fraud

A credit card allows us to borrow money from a recipient bank to make purchases. The issuer of the credit card has the condition that the cardholder will pay back the borrowed money with an additional agreed-upon charge.

Web Application - Injection

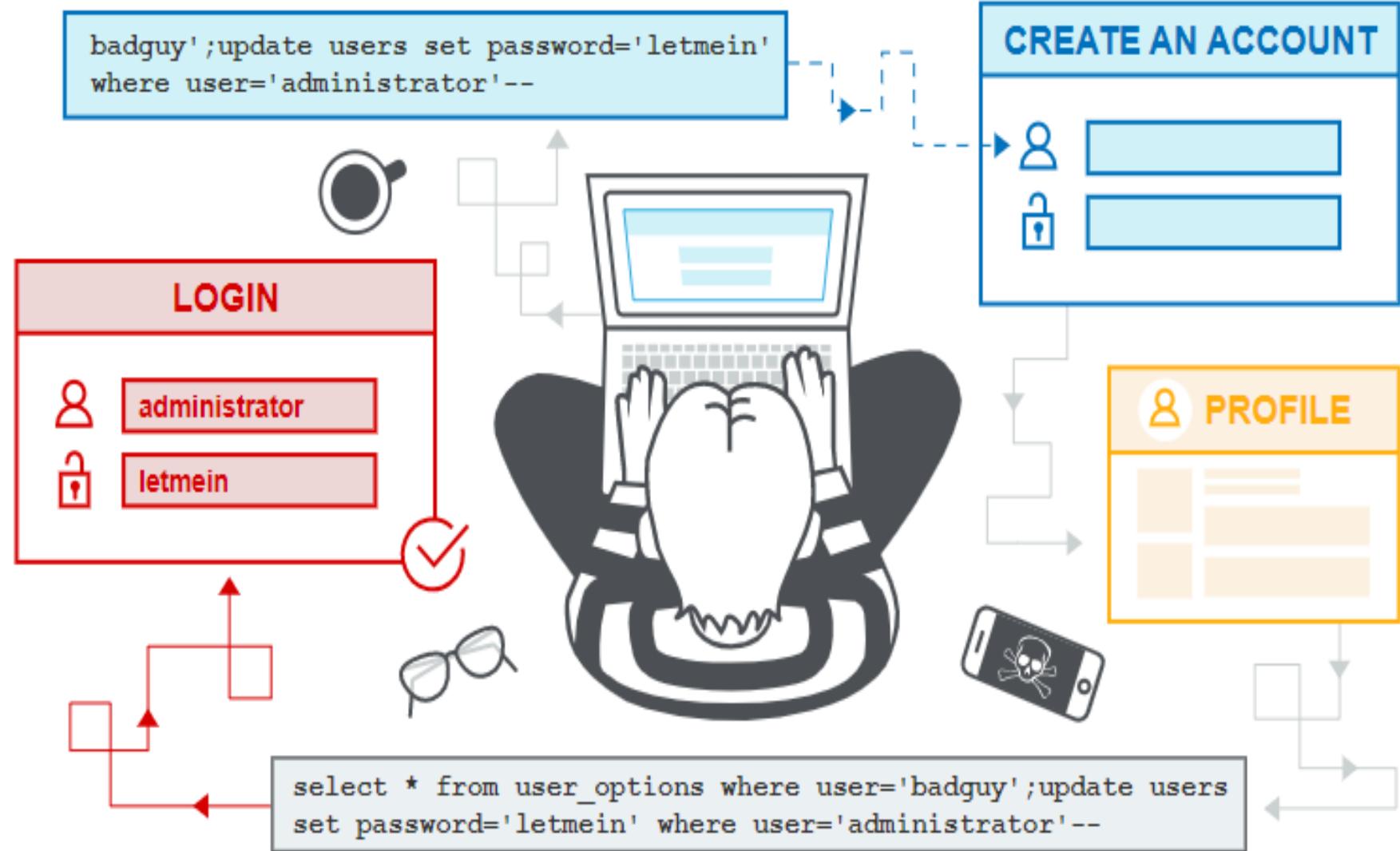
Injection technique consists of **injecting a SQL query** or a command using the input fields of the application.

A successful SQL injection can read, modify sensitive data from the database, and can also delete data from a database. It also enables the hacker to perform administrative operations on the database such as shutdown the DBMS/dropping databases.



Blind SQL injection is a type of SQL Injection attack that asks the database **true or false questions** and determines the answer based on the applications response. This attack is often used when the **web application is configured to show generic error messages**, but has not mitigated the code that is vulnerable to SQL injection.

Web Application - Injection



Web Application - Injection

```
String query = "SELECT * FROM EMP WHERE EMPID = '" + request.getParameter("id") + "'";
```

Step 1 – Navigate to the SQL Injection area of the application.

Step 2 – Here, we use String SQL Injection to **bypass authentication**. Use SQL injection to log in as the boss ('Neville') without using the correct password. Verify that Neville's profile can be viewed and that all functions are available (including Search, Create, and Delete).

Step 3 – We will **Inject a SQL** such that we are able to bypass the password by sending the parameter as 'a' = 'a' or 1 = 1. The SQL above is valid and will return ALL rows from the "EMP" table, since OR 1=1 is always TRUE.

Step 4 – **Post Exploitation**, we are able to login as Neville who is the Admin

Web Application - Injection

Preventing SQL Injection

There are plenty of ways to prevent SQL injection.

When developers write the code, they should ensure that they handle special characters accordingly. There are cheat sheets/prevention techniques available from **OWASP(Open Web Application Security Project)** which is definitely a guide for developers.

- Using Parameterized Queries
- Escaping all User Supplied Input
- Enable Least Privilege for the database for the end users

Web Application - Injection

```
String query = "SELECT * FROM EMP WHERE EMPID = '" + request.getParameter("id") + "'";
```

Burp Suite Free Edition v1.6

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts

Intercept HTTP history WebSockets history Options

Request to http://localhost:8080 [127.0.0.1]

Forward Drop Intercept is on Action

Raw Params Headers Hex

```
POST /WebGoat/attack?Screen=158&menu=1100 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Length: 43
Cookie: JSESSIONID=BD1C2E1DBF08A4359A6D4ED37575AD9F
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

employee_id=112&password=xyz' or 'a'='a'&action>Login
```

Web Application - Injection

```
String query = "SELECT * FROM EMP WHERE EMPID = '" + request.getParameter("id") + "'";
```

The screenshot shows a web browser window for 'Goat Hills Financial Human Resources'. The title bar says 'Welcome Back Neville - Staff Listing Page'. The main content area displays a list of staff members with their roles:

- Larry Stooge (employee)
- Moe Stooge (manager)
- Curly Stooge (employee)
- Eric Walker (employee)
- Tom Cat (employee)
- Jerry Mouse (hr)
- David Giambi (manager)
- Bruce McGuirre (employee)
- Sean Livingston (employee)
- Joanne McDougal (hr)
- John Wayne (admin)

To the right of the list are several buttons:

- SearchStaff
- ViewProfile
- CreateProfile
- DeleteProfile
- Logout

Free Source Code Analysers

- OWASP Orizon
- OWASP O2
- SearchDiggity
- FXCOP
- Splint
- Boon
- W3af
- FlawFinder

Commercial Source Code Analysers

- Parasoft C/C++ test
- HP Fortify
- Appscan
- Armorize CodeSecure
- GrammaTech



Are you a software Engineer ?



Thanks to All