Modelling Requirements

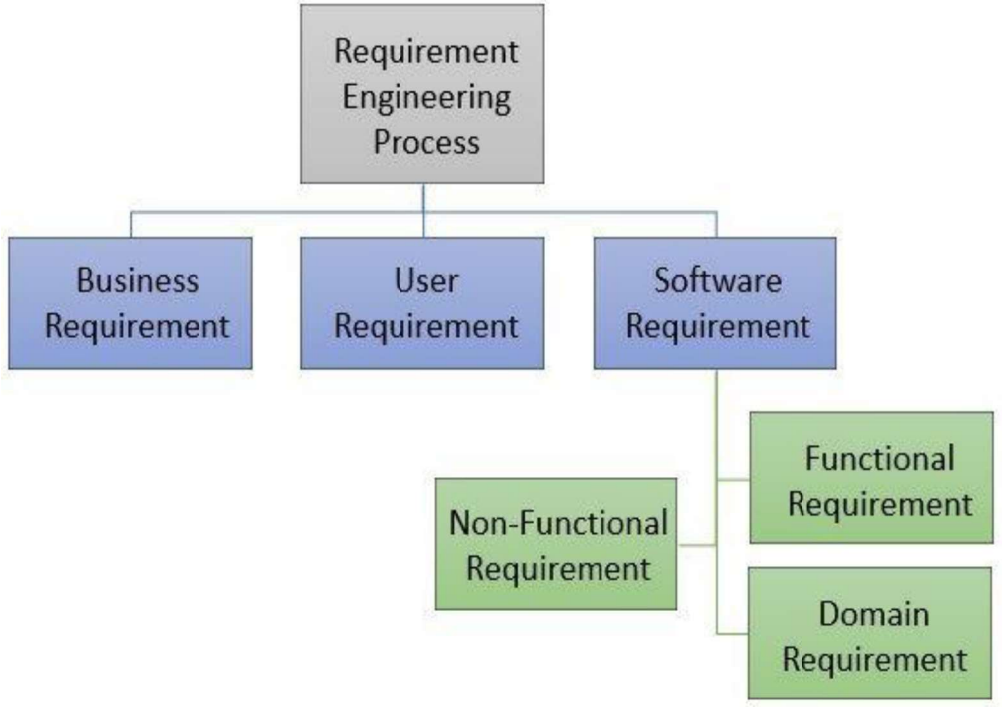
Software requirements and its types, Requirements Engineering process, Requirement Elicitation, System Modeling – Requirements Specification and Requirement Validation, Requirements Elicitation techniques, Requirements management in Agile.

Software requirements and its types

Types of Software Requirement

There are three types of software requirements as follows:

1. Functional requirements
2. Non-Functional requirements
3. Domain requirements



Functional Requirements

- Functional requirements are such software requirements that are demanded explicitly as basic facilities of the system by the end-users. So, these requirements for functionalities should be necessarily incorporated into the system as a part of the contract.
- They describe system behavior under specific conditions. In other words, they are the functions that one can see directly in the final product, and it was the requirements of the users as well. It describes a software system or its components.
- These are represented as inputs to the software system, its behavior, and its output. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

Functional software requirements help us to capture the intended behavior of the system.

Functional requirements can be incorporated into the system in many ways as

1. Natural language

2. A structured or formatted language with no rigorous syntax and formal specification language with proper syntax.

Examples of functional requirements

1. Whenever a user logs into the system, their authentication is done.

2. In case of a cyber attack, the whole system is shut down

3. Whenever a user registers on some software system the first time, a verification email is sent to the user.

Non-functional Requirements(NFRs)

- These requirements are defined as the quality constraints that the system must satisfy to complete the project contract.

Non-functional requirements are more abstract. They deal with issues like-

- Performance
- Reusability
- Flexibility
- Reliability
- Maintainability
- Security
- Portability

Non-Functional Requirements are classified into many types. Some of them are as:

- Interface Constraints
- Economic Constraints
- Operating Constraints
- Performance constraints**:** storage space, response time, security, etc.
- Life Cycle constraints: portability, maintainability, etc.

Domain Requirements

- Domain requirements are the requirements related to a particular category like software, purpose or industry, or other domain of projects. Domain requirements can be functional or non-functional. These are essential functions that a system of specific domains must necessarily exhibit.
- The common factor for domain requirements is that they meet established standards or widely accepted feature sets for that category of the software project. Domain requirements typically arise in military, medical, and financial industry sectors. They are identified from that specific domain and are not user-specific.

Examples of domain requirements are- medical equipment or educational software.

Software in medical equipment

- In medical equipment, software must be developed per IEC 60601 regarding medical electrical equipment's basic safety and performance.
- The software can be functional and usable but not acceptable for production because it fails to meet domain requirements.

An Academic Software

- Such software must be developed to maintain records of an institute efficiently.
- Domain requirement of such software is the functionality of being able to access the list of faculty and list of students of each grade.

- **Difference between Functional Requirement and Non-Functional Requirement**

Functional Requirement	Non-Functional Requirement
It is used for defining a system and its components.	It is used for defining the quality attribute of a software system.
It focuses on what software will be doing.	It fixes the constraint on which software should fulfill the functional requirement.
The user specifies it.	Techies like architects or software developers specify it.
It is compulsory.	It is not compulsory.
It is easy to define.	It is comparatively tough to define.
It verifies the functionality of the system.	It verifies the performance of the system.
It is defined as a system at the component level.	It is defined as a system as a whole.
Example-System should be shut down if a cyber attack happens.	Example-Within 10 seconds, the processing should be done of each request.

Requirements Engineering process

- **Requirements engineering (RE)** refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system.

Requirement Engineering Process is a five-step process, which includes -

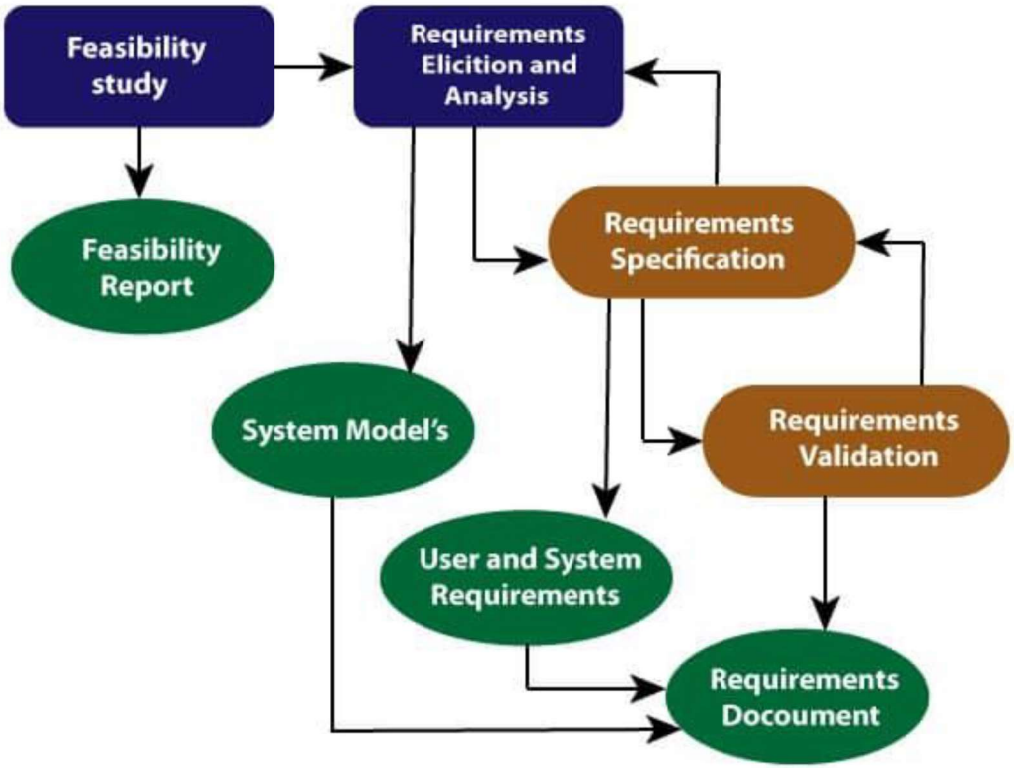
1.Feasibility Study

2.Requirement Elicitation and Analysis

3.Software Requirement Specification

4.Software Requirement Validation

5.Software Requirement Management



Requirement Engineering Process

1. Feasibility Study:

- The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

Types of Feasibility:

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

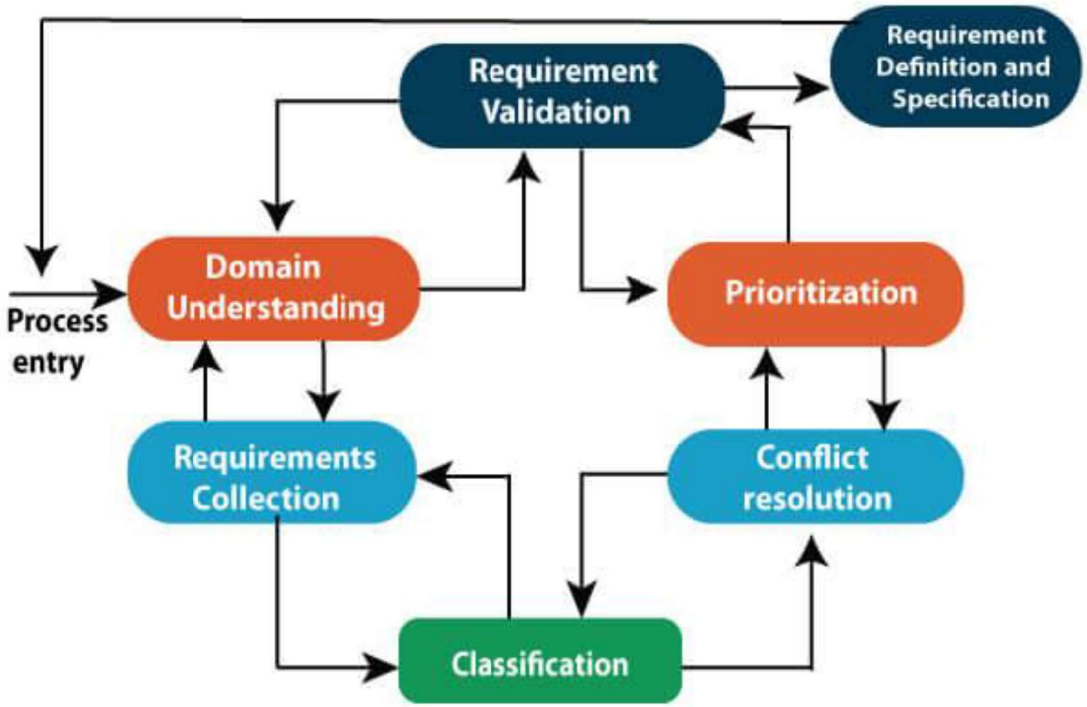
2.Requirement Elicitation and Analysis:

- This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.
- Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of Elicitation and Analysis

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements.

Elicitation and Analysis Process



3. Software Requirement Specification:

- Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.
- The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.
- **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.
- **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
- **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "***E-R diagram***." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

4. Software Requirement Validation:

- After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions -
- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirements Validation Techniques

- **Requirements reviews/inspections:** systematic manual analysis of the requirements.
- **Prototyping:** Using an executable model of the system to check requirements.
- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

5. Software Requirement management

A typical requirements management process complements the through these steps:

- Collect initial requirements from stakeholders
- Analyze requirements
- Define and record requirements
- Prioritize requirements
- Agree on and approve requirements
- Trace requirements to work items
- Query stakeholders after implementation on needed changes to requirements
- Utilize test management to verify and validate system requirements
- Assess impact of changes
- Revise requirements
- Document changes

Some of the benefits of requirements management include:

- Lower cost of development across the lifecycle
- Fewer defects
- Minimized risk for safety-critical products
- Faster delivery
- Reusability
- Traceability
- Requirements being tied to test cases
- Global configuration management

Requirements Elicitation

- Requirement elicitation can be done by communicating with stakeholders directly or by doing some research, experiments. The activities can be planned, unplanned, or both.
- **Planned activities** include workshops, experiments.
- **Unplanned activities** happen randomly. Prior notice is not required for such activities. **For example**, you directly go to the client site and start discussing the requirements however there was no specific agenda published in advance.

Following tasks are the part of elicitation:

- **Prepare for Elicitation:** The purpose here is to understand the elicitation activity scope, select the right techniques, and plan for appropriate resources.
- **Conduct Elicitation:** The purpose here is to explore and identify information related to change.
- **Confirm Elicitation Results:** In this step, the information gathered in the elicitation session is checked for accuracy.

Requirements Elicitation Techniques

- **There are several techniques available for elicitation, however, the commonly used techniques are explained below:**

1) Stakeholder Analysis

- Stakeholders can include team members, customers, any individual who is impacted by the project or it can be a supplier. Stakeholder analysis is done to identify the stakeholders who will be impacted by the system.

2) Brainstorming

- This technique is used to generate new ideas and find a solution for a specific issue. The members included for brainstorming can be domain experts, subject matter experts.

- **Brainstorming technique is used to answer the below questions:**
- What is the expectation of a system?
- What are the risk factors that affect the proposed system development and what to do to avoid that?
- What are the business and organization rules required to follow?
- What are the options available to resolve the current issues?
- What should we do so that this particular issue does not happen in the future?

3) Interview

Interview techniques should be used for building strong relationships between business analysts and stakeholders.

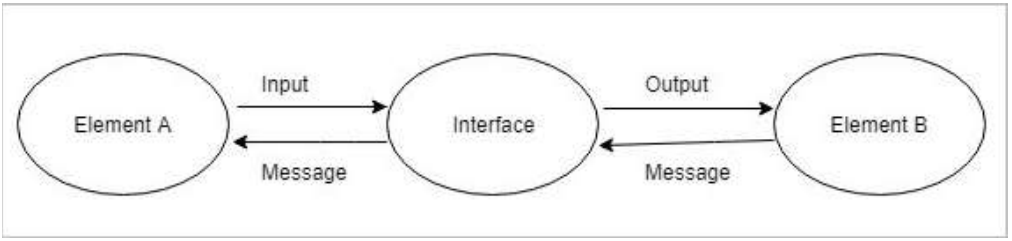
- If the interviewer has a predefined set of questions then it's called **a structured interview.**
- If the interviewer is not having any particular format or any specific questions then it's called an **unstructured interview**.

4) Document Analysis/Review

This technique is used to gather business information by reviewing/examining the available materials that describe the business environment. This analysis is helpful to validate the implementation of current solutions and is also helpful in understanding the business need.

- Document analysis includes reviewing the business plans, technical documents, problem reports, existing requirement documents, etc. This is useful when the plan is to update an existing system. This technique is useful for migration projects.
- 5) Focus Group
- The Focus group includes subject matter experts. The objective of this group is to discuss the topic and provide information. A moderator manages this session.
- The moderator should work with business analysts to analyze the results and provide findings to the stakeholders.
- If a product is under development and the discussion is required on that product then the result will be to update the existing requirement or you might get new requirements. If a product is ready to ship then the discussion will be on releasing the product.

- 6) Interface Analysis
- Interface analysis is used to review the system, people, and processes. This analysis is used to identify how the information is exchanged between the components. An Interface can be described as a connection between two components. **This is described in the below image:**



The interface analysis focus on the below questions:

1.Who will be using the interface?
2.What kind of data will be exchanged?
3.When will the data be exchanged?
4.How to implement the interface?
5.Why we need the interface? Can't the task be completed without using the interface?

- 7) Observation
- The main objective of the observation session is to understand the activity, task, tools used, and events performed by others.
- The plan for observation ensures that all stakeholders are aware of the purpose of the observation session, they agree on the expected outcomes, and that the session meets their expectations
- During the session, the observer should record all the activities and the time taken to perform the work by others so that he/she can simulate the same. Observation can be either active or passive.
- **Active observation** is to ask questions and try to attempt the work that other persons are doing.
- **Passive observation** is silent observation i.e. you sit with others and just observe how they are doing their work without interpreting them.

- 8) Prototyping
- Prototyping is used to identify missing or unspecified requirements. In this technique, frequent demos are given to the client by creating the prototypes so that client can get an idea of how the product will look like.
- 9) Joint Application Development (JAD)/ Requirement Workshops
- This technique is more process-oriented and formal as compared to other techniques. These are structured meetings involving end-users. This is used to define, clarify, and complete requirements.
- **This technique can be divided into the following categories:**
- **Formal Workshops:** These workshops are highly structured and are usually conducted with the selected group of stakeholders. The main focus of this workshop is to define, create, refine, and reach closure on business requirements.
- **Business Process Improvement Workshops:** These are less formal as compared to the above one. Here, existing business processes are analyzed and process improvements are identified.

- 10) Survey/Questionnaire
- For Survey/Questionnaire, a set of questions is given to stakeholders to quantify their thoughts. After collecting the responses from stakeholders, data is analyzed to identify the area of interest of stakeholders.
- Questions should be based on high priority risks. Questions should be direct and unambiguous. Once the survey is ready, notify the participants and remind them to participate.
- **Two types of questions can be used here:**
- **Open-Ended:** Respondent is given the freedom to provide answers in their own words rather than selecting from predefined responses. This is useful but at the same time, this is time-consuming as interpreting the responses is difficult.
- **Close Ended:** It includes a predefined set of answers for all the questions and the respondent has to choose from those answers. Questions can be multiple choice or can be ranked from not important to very important.

System Modeling

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- most models use some kind of graphical notation representing a system, which is known as Unified Modeling Language (UML)

Benefits of System Modeling

1. Ease project management tasks.
2. Can provide complete views of a system, as well as detailed views of subsystems.
3. Clarify structures and relationships.
4. Offer a communication framework for ideas within and between teams. Can generate new ideas and possibilities.
5. Allow quality assurance and testing scenarios to be generated.
6. Platform independent.

UML Diagram Types

- The System Perspectives (last slide) are modeled with diagrams
- Activity diagrams, which show the activities involved in a process or in data processing .
- Use case diagrams, which show the interactions between a system and its environment.
- Sequence diagrams, which show interactions between actors and the system and between system components.
- Class diagrams, which show the object classes in the system and the associations between these classes.
- State diagrams, which show how the system reacts to internal and external events.

Types of Models

1. Context models
2. Interaction models
3. Structural models
4. Behavioral models
5. Model-driven engineering

1. Context Models

- Context models are used to illustrate the operational context of a system -
- They show what lies outside the system boundaries.

- Social and organizational concerns may affect the decision on where to position system boundaries.

- Architectural models show the system and its relationship with other systems.

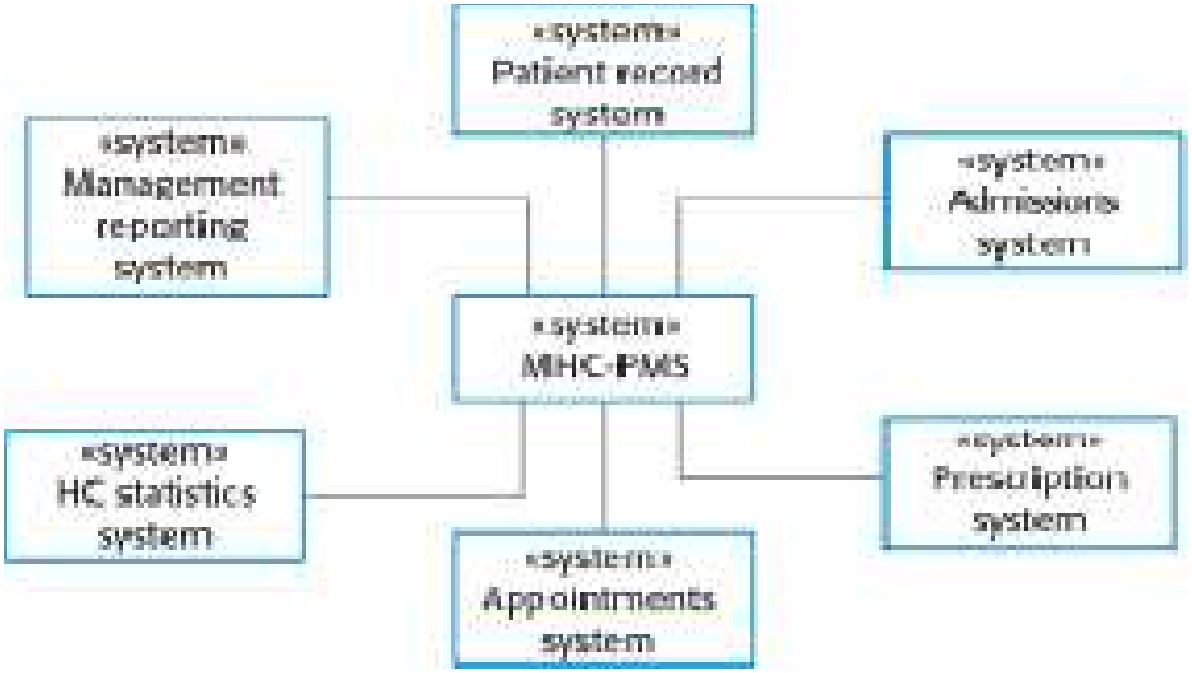
System Boundaries

- System boundaries are established to define what is inside and what is outside the system.
  - They show other systems that are used or depend on the system being developed.

- The position of the system boundary has a profound effect on the system requirements.

- Defining a system boundary is a political judgment
  - There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

The Context of the MHC-PMS



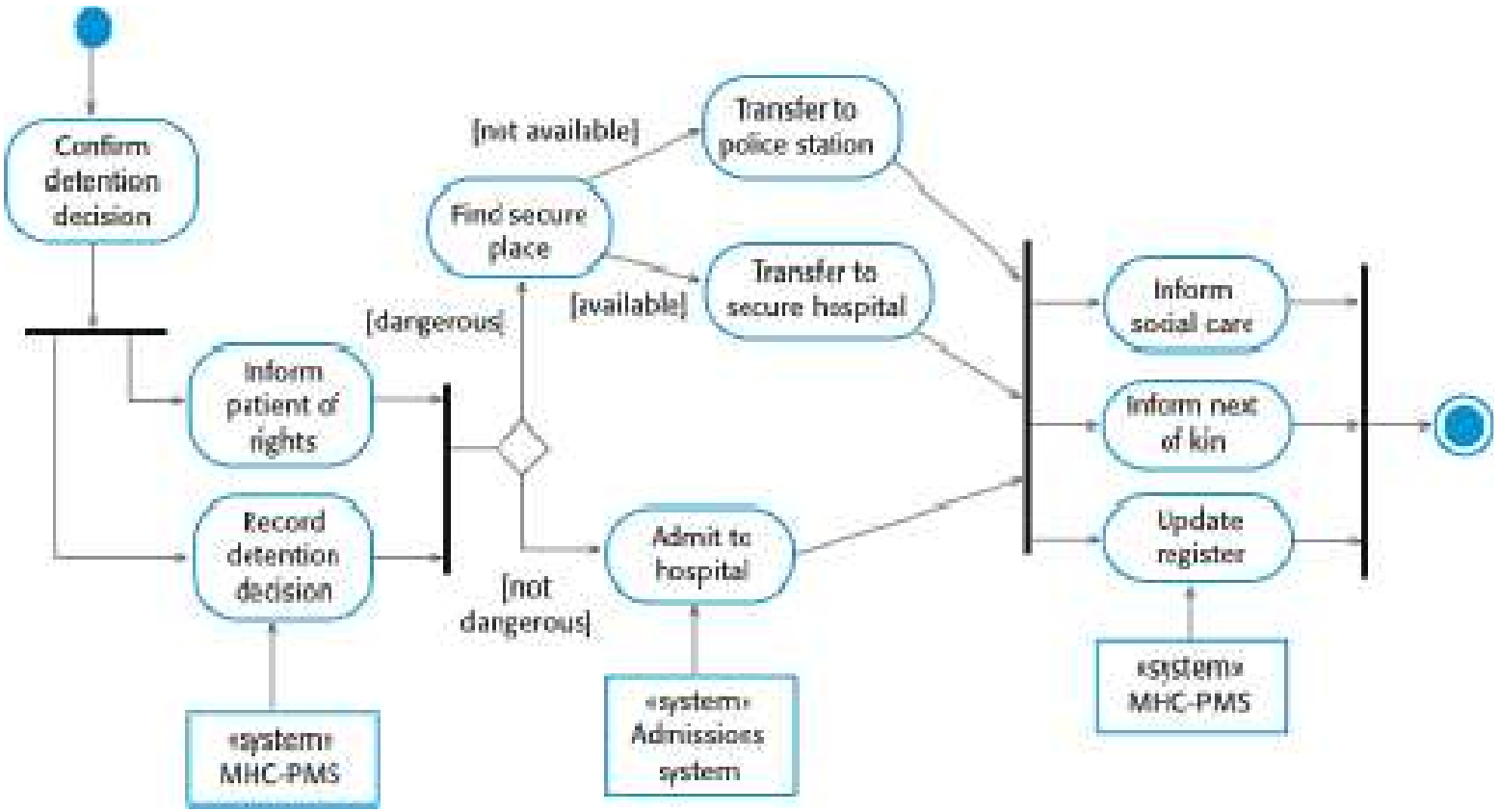
Process Perspective

- Context models simply show the other systems in the environment, not how the system being developed is used in that environment.

- Process models reveal how the system being developed is used in broader business processes.
- How it 'works' Detailed.

- UML activity diagrams may be used to define business process models.

Process Model of Involuntary Detention



2. Interaction Models

- Modeling user interaction is used to identify user requirements.
- We see structural connections and dynamic (behavioral) interactions.
- We do this with graphical models.

- Use case diagrams and sequence diagrams may be used for interaction modeling.
- These are the most popular modeling mechanisms

Use Case Modeling (Interaction Model)

- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.
- Each use case represents a discrete task that involves external interaction with a system.
- Actors in a use case may be people, devices, or other systems.
- Represented diagramatically to provide an overview of the use case and in a more detailed textual form.

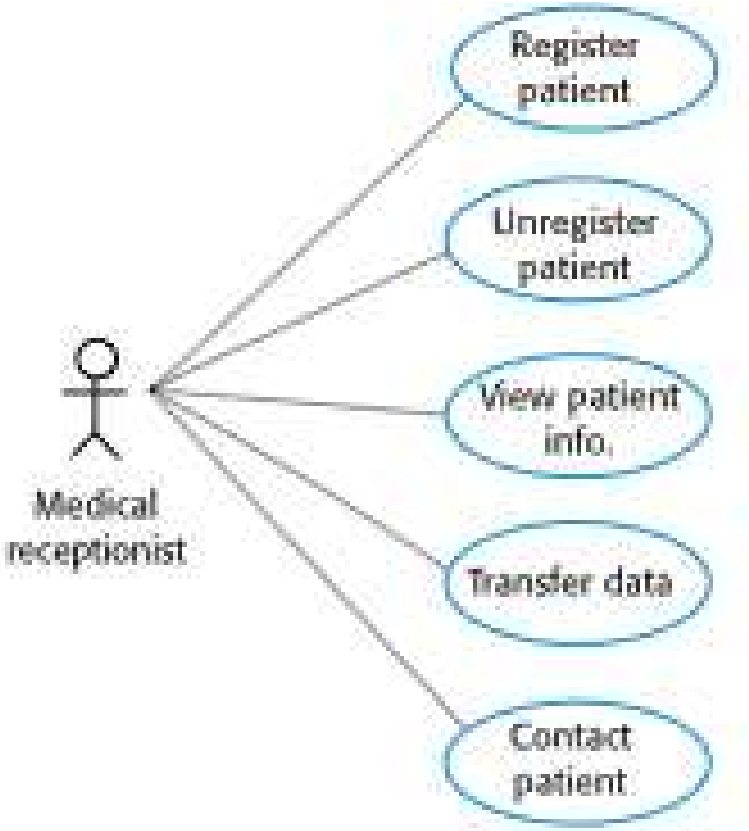
Transfer-data Use Case Diagram (graphical model)

❿A use case in the MHC-PMS

Tabular Description of the ‘Transfer data’ use-case

MHC-PMS: Transfer data	
Actors	Medical receptionist, patient records system (PRS)
Description	A receptionist may transfer data from the MHC-PMS to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient’s diagnosis and treatment.
Data	Patient’s personal information, treatment summary
Stimulus	User command issued by medical receptionist
Response	Confirmation that PRS has been updated
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.

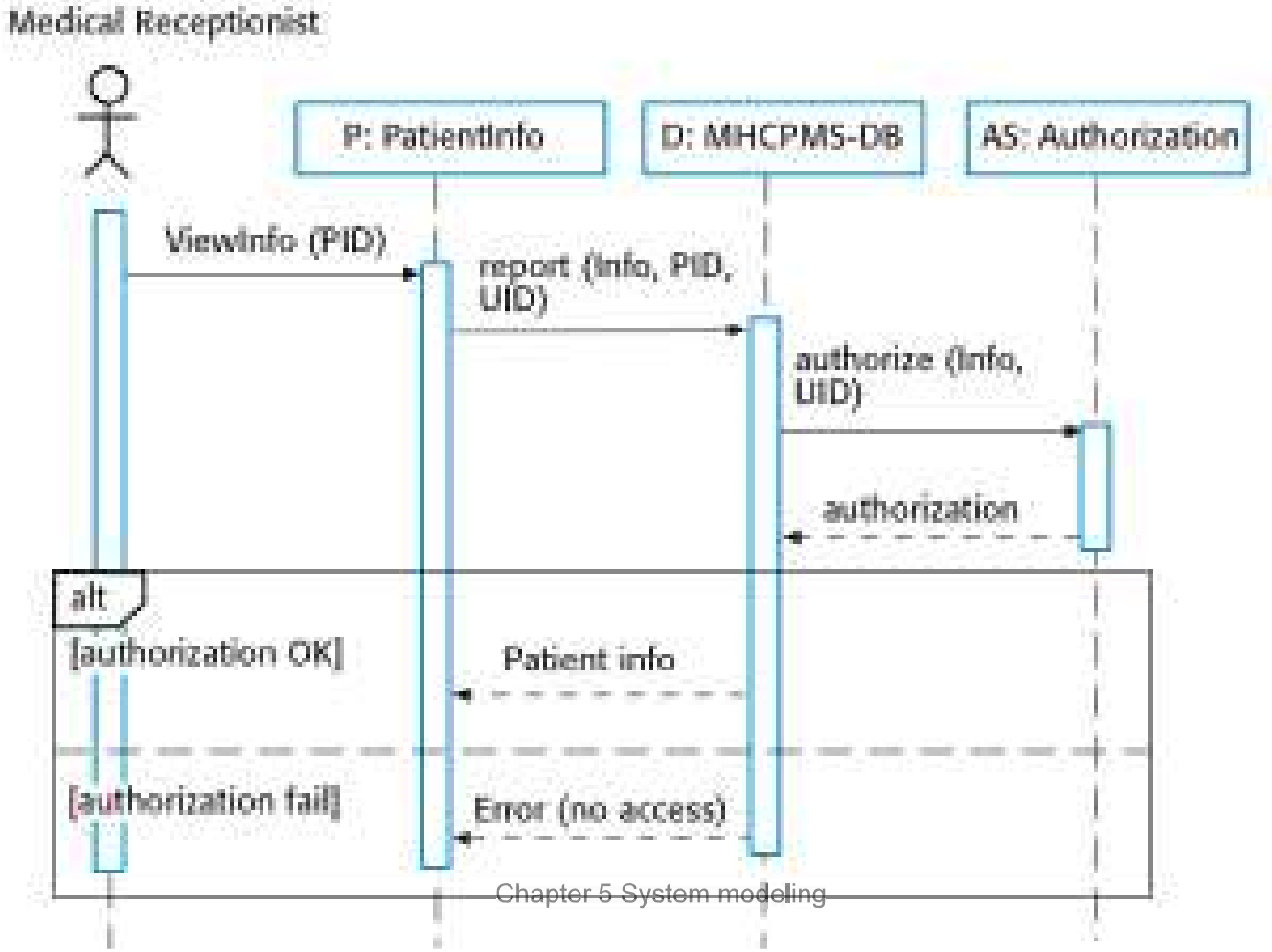
Use Cases in the MHC-PMS involving the role ‘Medical Receptionist’ (only showing one actor here)



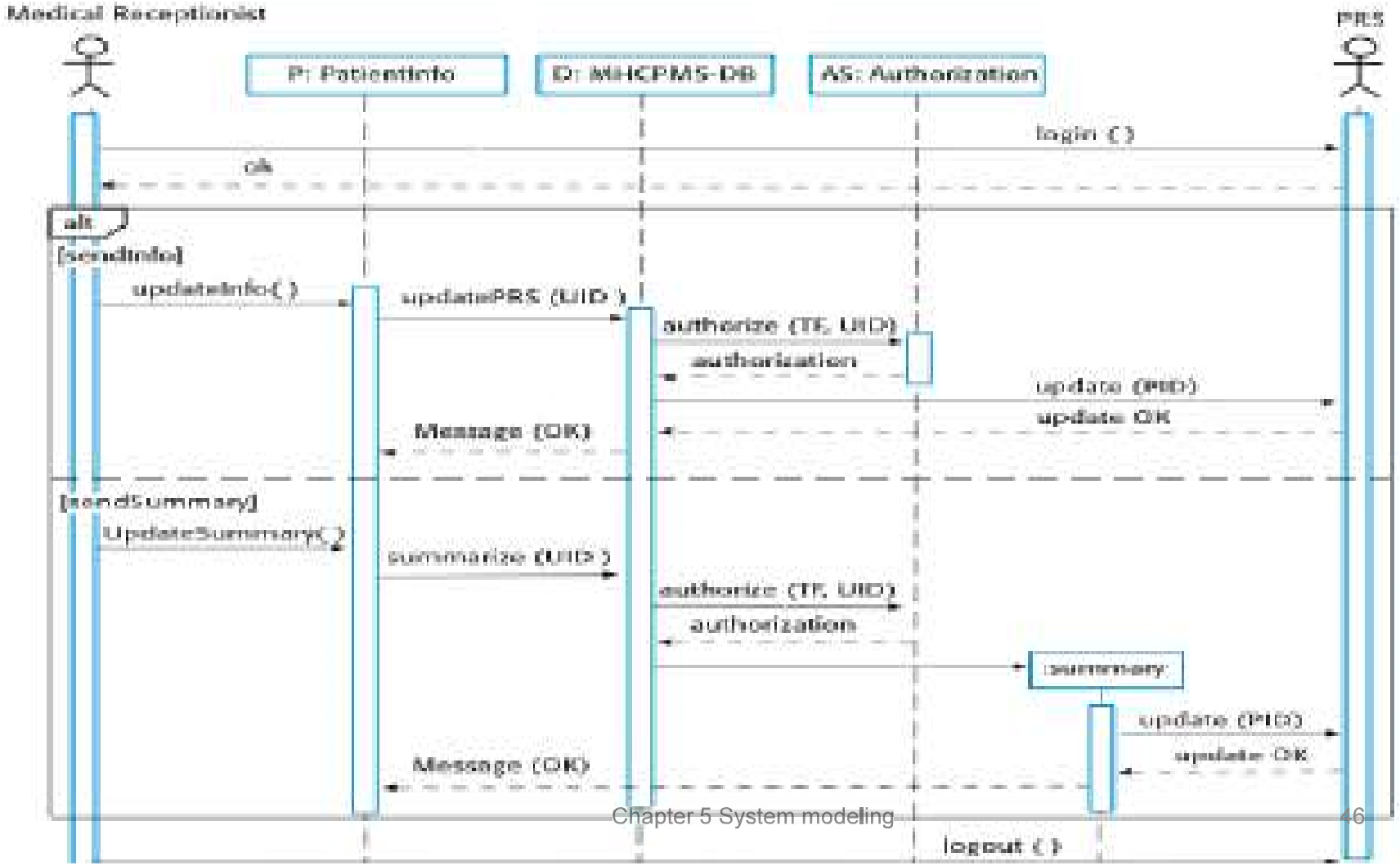
Sequence Diagrams (Interaction Model)

- Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system.
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- Interactions between objects are indicated by annotated arrows.

Sequence diagram for View Patient Information



Sequence diagram for Transfer Data



Key points

- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behavior.

- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.

- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.

Requirements Specification and Requirement Validation

- For requirement specification write(SRS)

Once the SRS completed validation begins

Requirement Validation

- The development of software begins once the requirements document is ‘ready’
- Ensure that the requirements specification (SRS)contains no errors and that it specifies the user’s requirements correctly.
- If errors present in the SRS will adversely affect the cost if they are detected later in the development process or when the software is delivered to the user.

- So it is desirable to detect errors in the requirements before the design and development of the software begins.
- In the validation phase the requirements are examined for consistency, omissions, and ambiguity.
- The basic objective is to ensure that the SRS reflects the actual requirements accurately and clearly.
- objectives of the requirements document are given below.

1. To certify that the SRS contains an acceptable description of the system to be implemented
2. To ensure that the actual requirements of the system are reflected in the SRS
3. To check the requirements document for completeness, accuracy, consistency, requirement conflict', conformance to standards and technical errors.

- Requirements validation studies the 'final draft' of the requirements document while requirements analysis studies the 'raw requirements' from the system stakeholders (users).

- **Requirements validation**: Have we got the requirements right?
- **Requirements analysis**: Have we got the right requirements?

- The requirements document should be formulated and organized according to the standards of the organization.
- The **organizational standards are** specified standards followed by the organization according to which the system is to be developed.

Requirements Validation

The **agreed actions** is a list that displays the actions to be performed to resolve the problems depicted in the problem list.

Requirements Review

- Requirements validation determines whether the requirements are substantial to design the system.

The problems encountered during requirements validation are listed below.

1. Unclear stated requirements
2. Conflicting requirements are not detected during requirements analysis
3. Errors in the requirements elicitation and analysis
4. Lack of conformance to quality standards.

- To avoid the problems stated above, a **requirements review** is conducted, which consists of a review team that performs a systematic analysis of the requirements.
- The review team consists of software engineers, users, and other stakeholders who examine the specification to ensure that the problems associated with consistency, omissions, and errors are detected and corrected.

- In the review meeting check list are:

1. Is the initial state of the system defined?
2. Is there a conflict between one requirement and the other?
3. Are all requirements specified at the appropriate level of abstraction?
4. Is the requirement necessary or does it represent an add-on feature that may not be essentially implemented?
5. Is the requirement bounded and has a clear defined meaning?
6. Is each requirement feasible in the technical environment where the product or system is to be used?
7. Is testing possible once the requirement is implemented?
8. Are requirements associated with performance, behavior, and operational characteristics clearly stated?
9. Are requirements patterns used to simplify the requirements model?
10. Are the requirements consistent with the overall objective specified for the system/product?
11. Have all hardware resources been defined?
12. Is the provision for possible future modifications specified?
13. Are functions included as desired by the user (and stakeholder)?
14. Can the requirements be implemented in the available budget and technology?
15. Are the resources of requirements or any system model (created) stated clearly?

Other Requirements Validation Techniques

- A number of other requirements validation techniques are used either individually or in conjunction with other techniques to check the entire system or parts of the system. The selection of the validation technique depends on the appropriateness and the size of the system to be developed. Some of these techniques are listed below.

1. **Test case generation:** The requirements specified in the SRS document should be testable. The test in the validation process can reveal problems in the requirement. In some cases test becomes difficult to design, which implies that the requirement is difficult to implement and requires improvement.
2. **Automated consistency analysis:** If the requirements are expressed in the form of structured or formal notations, then CASE tools can be used to check the consistency of the system. A requirements database is created using a CASE tool that checks the entire requirements in the database using rules of method or notation. The report of all inconsistencies is identified and managed.
3. **Prototyping:** Prototyping is normally used for validating and eliciting new requirements of the system. This helps to interpret assumptions and provide an appropriate feedback about the requirements to the user. For example, if users have approved a prototype, which consists of graphical user interface, then the user interface can be considered validated.

Requirements Elicitation techniques

https://www.softwaretestinghelp.com/requirements-elicitation-techniques/

Requirements management in Agile

- Agile requirements management is built on the principle of flexibility, so any potential challenges are identified and resolved much earlier in the process, minimizing expensive and costly rework. A few benefits of the agile approach include:
- **Improved product design and delivery.** A recent report suggests that best-in-class RM solutions can significantly improve product design and delivery for agile development teams. "In the face of increasing regulations, connected products for the internet of things (IoT), and scaling Agile practices, AD&D [application development and delivery] leaders long for something to bring traceability and auditability to their processes without sacrificing speed."
- **Improved traceability.** The right RM solution can enhance development transparency through traceability. Traceability empowers teams to perform impact analysis more readily, which is critical to product development.
- **Achieve quicker time to market.** Teams are facing more complexity and pressure to comply with industry regulations, and need to measure customer value to search, track and connect interdependent requirements. Achieving faster time to market requires that teams collaborate faster and more effectively, working to build traceability requirements and test case

Designing with greater flexibility through the agile requirements management lifecycle

The agile requirements management lifecycle is focused on clearly defining a project's scope, so that you better understand what needs to happen to meet the desired end goal. It provides a high-level understanding of business goals, and outlines what is needed for the project to be a success.

Consider taking the following steps:

- **Understand user stories.** User stories give you powerful information about the problem you're trying to solve. A user story is a quick description of everyday use cases and might include a few sentences about how the user expects the product to perform. A template might be something like this: "As a [role], I need [product] to do [goal of the software] so that I can [benefit of the product]."
- **Outline the most important requirements.** Identify what requirements are most essential based on the high-level business strategy. These requirements may be supported by user stories, functional requirements and more based on the specific client goals.
- **Transform to product features.** This stage is about fine-tuning and translating the details that you've gathered into product features. The development team collaborates to ensure that any requirements are easily understood by anyone working to implement them. User stories are linked to features and tasks, so that developers understand what they need to do – but also why they are doing it.

- **Collaboration and Buy-In**
- It’s difficult to get a company to agree on requirements, particularly for large projects with many stakeholders. In practice, it’s not necessary to achieve consensus through compromise. It’s more important to have team buy-in (before or after management approves the project) so the development process can move forward. With buy-in, the team backs the best solution, makes a smart decision and does what is necessary to move forward with the requirements management process.
- Team collaboration is key to establishing good requirements. Collaborative teams work hard to make sure everyone has a stake in the project and provides feedback. When there is a commitment and understanding of project goals, team members tend to support other’s decisions. It’s when developers, testers or other stakeholders feel “out of the loop” that communication issues arise, people get frustrated and projects get delayed.

- **Traceability and Change Management**
- Requirements traceability is a way to keep everyone in the loop. It organizes, documents and keeps track of all requirements, from initial idea to testing. A simple metaphor for traceability is connecting the dots to identify the relationships between items within a project. The following figure shows an example of a common downstream flow.

- Companies should be able to trace each requirement back to its original business objective throughout the development process, not merely after it’s complete. By tracing requirements, companies can identify the ripple effect changes have, see if they have completed a requirement and if they tested it properly. With traceability, and by managing changes effectively, managers gain the visibility to anticipate issues and ensure continuous quality.
- Traceability also makes sure the product meets all the vital requirements that come from different stakeholders. By tracing requirements, all team members stay connected to each other and to all interdependencies. And by managing changes well, a company can avoid scope creep — unplanned changes that occur when requirements are not clearly captured, understood and communicated. The benefit of good requirements is a clear understanding of the product and the scope involved. This leads to a better development schedule and budget, which prevents delays and cost overruns.

- **Quality Assurance**
- Getting requirements right the first time means better quality, faster development cycles and higher customer satisfaction with the product. Concise, specific requirements can help companies detect and solve problems earlier rather than later, when they are much more expensive to fix.
- Research has shown that project teams can eliminate 50 percent to 80 percent of project defects by effectively managing requirements. In addition, according to Borland Software (now Micro Focus), it can cost up to 100 times more to correct a defect later in the development process, after it's been coded, than when it's still in written form.
- By integrating requirements management best practices into their quality assurance process, companies can help teams increase efficiency and eliminate rework. According to the Carnegie Mellon Software Engineering Institute, 60 to 80 percent of the cost of software development is in rework. In other words, development teams are wasting the majority of their budgets on efforts they didn't perform correctly the first time.
- Requirements management best practices can appear to be a complex topic, but at its core, it's a simple concept. It helps teams answer the question: Does everyone — from business leaders to product managers and project leaders to developers, QA managers and testers — understand what is being built and why?
- When everyone is collaborating and has full context and visibility into the discussions, decisions and changes involved in product development, they maintain high quality and almost always ensure success.

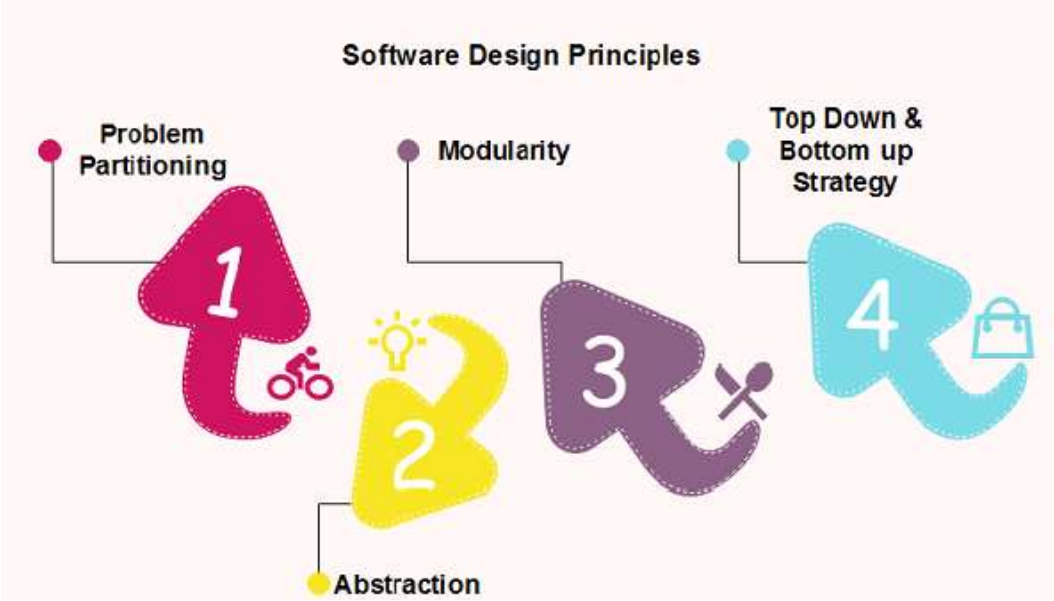
Software Design

Design concepts and principles - Abstraction - Refinement - Modularity Cohesion coupling, Architectural design, Detailed Design Transaction Transformation, Refactoring of designs, Object oriented Design User-Interface Design

Software Design concepts and principles

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

principles of Software Design



- 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.

For software design, the goal is to divide the problem into manageable pieces.

- 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

Abstraction

- An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.
- Here, there are two common abstraction mechanisms

1. Functional Abstraction
2. Data Abstraction

Functional Abstraction

i. A module is specified by the method it performs.

- The details of the algorithm to accomplish the functions are not visible to the user of the function. Functional abstraction forms the basis for **Function oriented design approaches**.

Data Abstraction

- Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.

Modularity

- Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

The desirable properties of a modular system are:

- Each module is a well-defined system that can be used with other applications.
- Each module has single specified objectives.
- Modules can be separately compiled and saved in the library.
- Modules should be simple and easier to use.

Advantages of Modularity

- It allows large programs to be written by several or different people
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It simplifies the overlay procedure of loading a large program into main storage.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test
- It produced the well designed and more readable program.

Disadvantages of Modularity

- Execution time maybe longer
- Storage certainly increased
- Compilation and loading time may be longer
- Inter-module communication problems may be increased
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

Modular Design

- Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system.
- **1. Functional Independence:** Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules. Independence is important because it makes implementation more accessible and faster. The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well. Thus, functional independence is a good design feature which ensures software quality.

It is measured using two criteria:

- **Cohesion:** It measures the relative function strength of a module.
- **Coupling:** It measures the relative interdependence among modules.

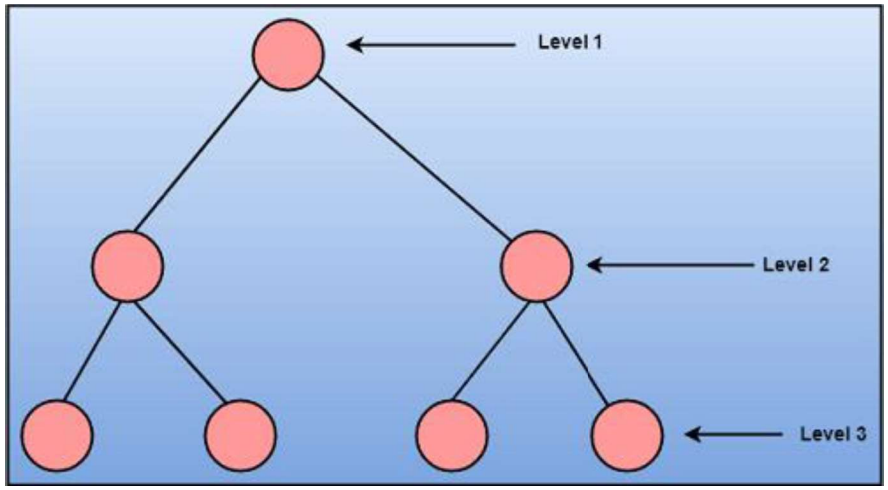
- **2.Information hiding:** The fundamental of Information hiding suggests that modules can be characterized by the design decisions that protect from the others, i.e., In other words, modules should be specified that data include within a module is inaccessible to other modules that do not need for such information.
- The use of information hiding for modular system provides the most significant benefits when modifications are required during testing's and later during software maintenance. This is because as most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to different locations within the software.

Strategy of Design

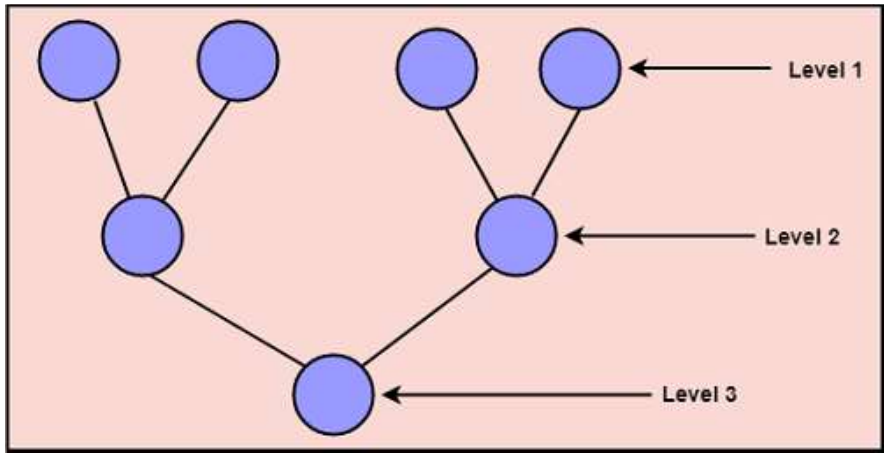
- A good system design strategy is to organize the program modules in such a method that are easy to develop and to change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.
- To design a system, there are two possible approaches:

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

- **1. Top-down Approach:** This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



2. Bottom-up Approach: A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.

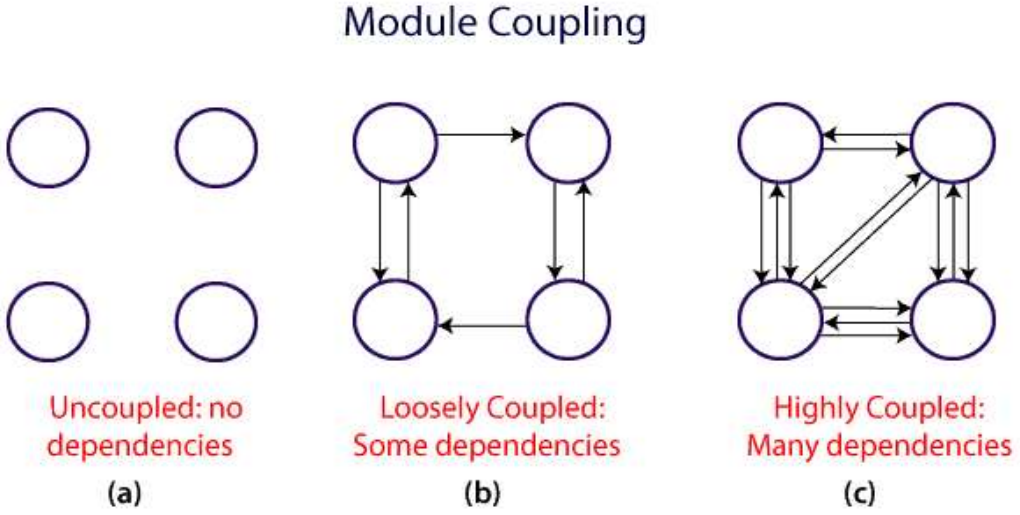


Coupling and Cohesion

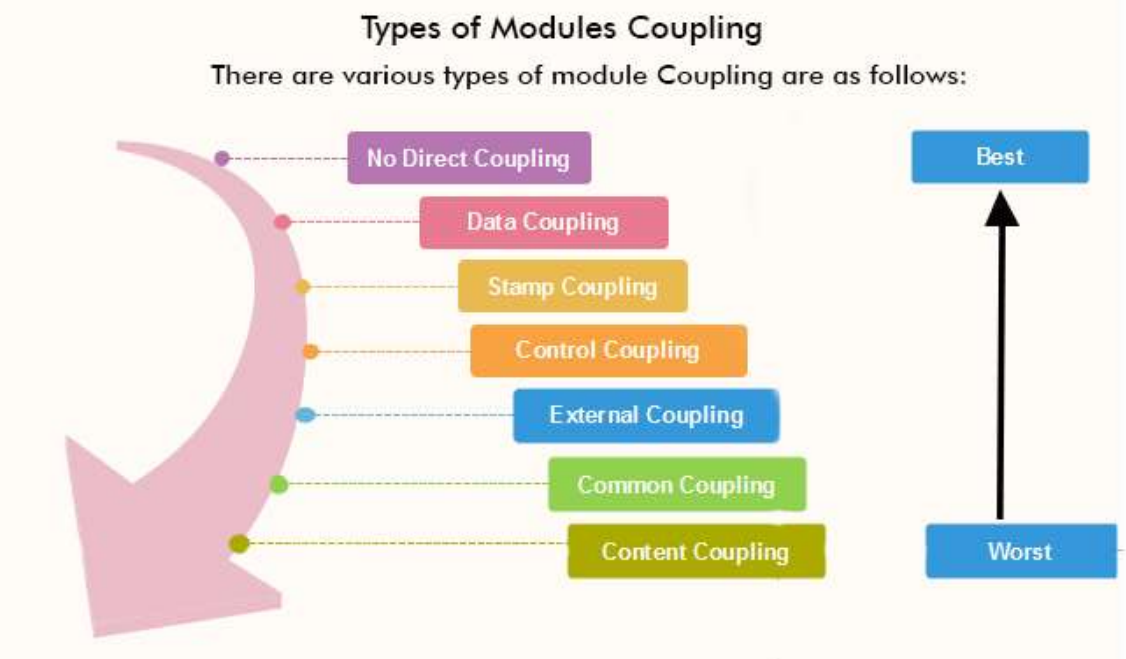
- Module Coupling

- In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all within them.

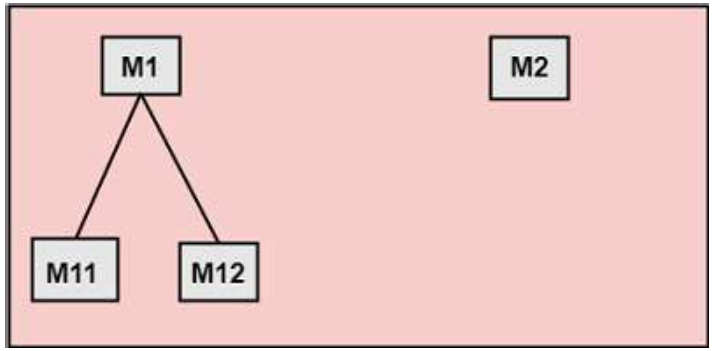
The various types of coupling techniques are shown in fig



- A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more errors.

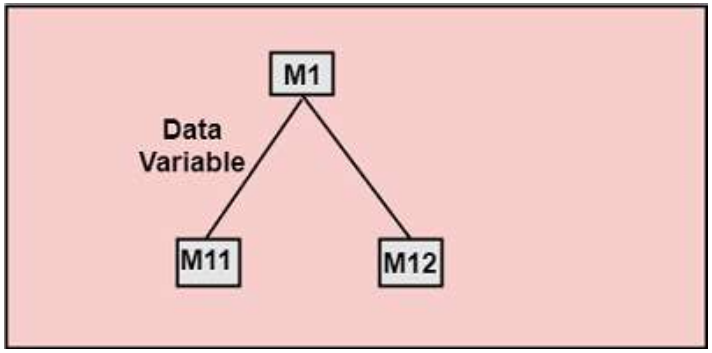


1. No Direct Coupling: There is no direct coupling between M1 and M2.

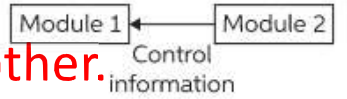


In this case, modules are subordinates to different modules. Therefore, no direct coupling.

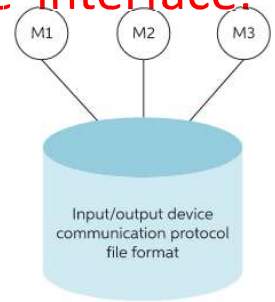
2. Data Coupling: When data of one module is passed to another module, this is called data coupling.



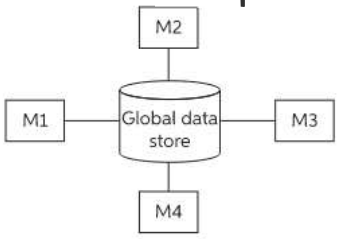
- **3.Stamp Coupling:** Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.
- **4. Control Coupling:** Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

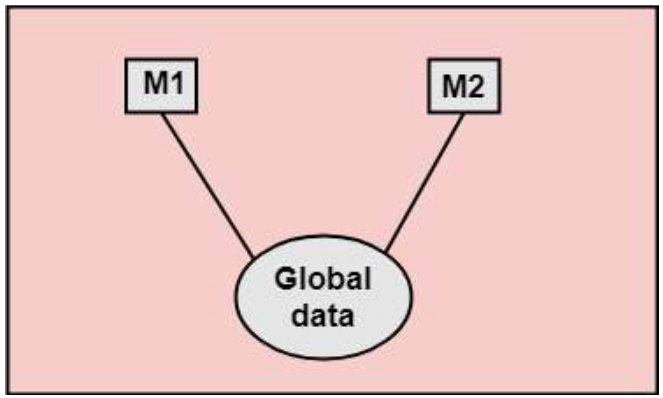


- **5. External Coupling:** External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.



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



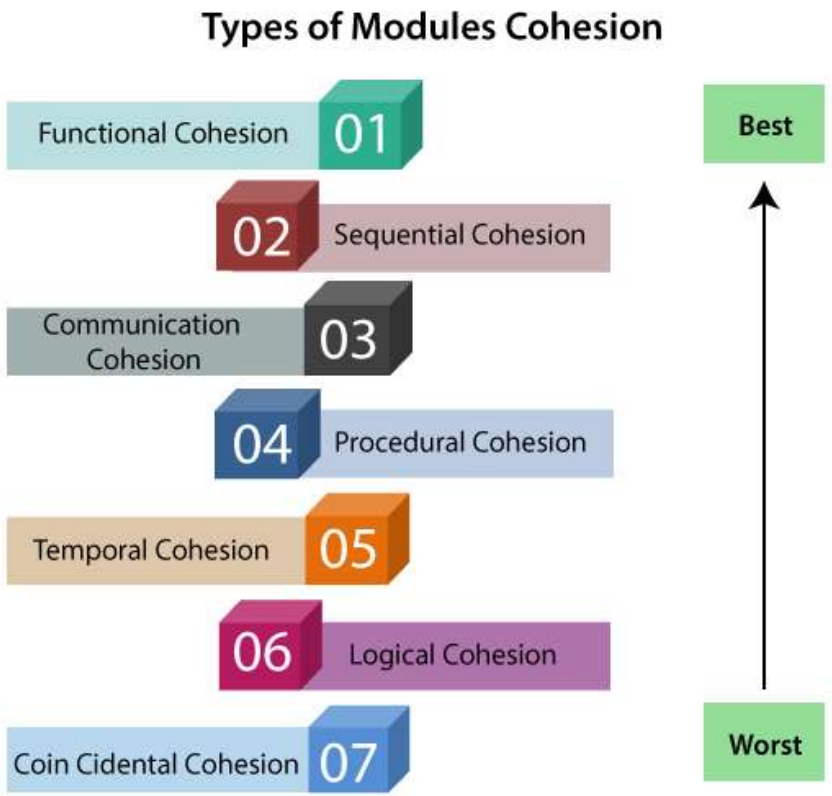


7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

Module Cohesion

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

Types of Modules Cohesion



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

example of functional cohesion includes reading transaction records because all the elements related to single transaction are involved in it.

2. Sequential Cohesion: A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.

3.Communicational Cohesion: A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.

4.Procedural Cohesion: A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.

5.Temporal Cohesion: When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.

6.Logical Cohesion: A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.

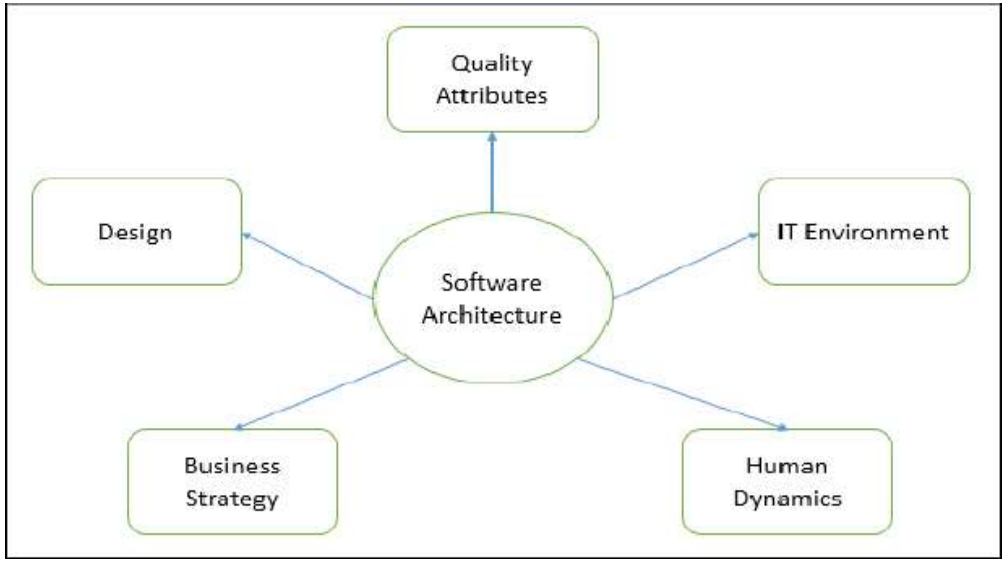
7.Coincidental Cohesion: A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely.

- Differentiate between Coupling and 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.

Software Architecture Design

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



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

Software Architecture

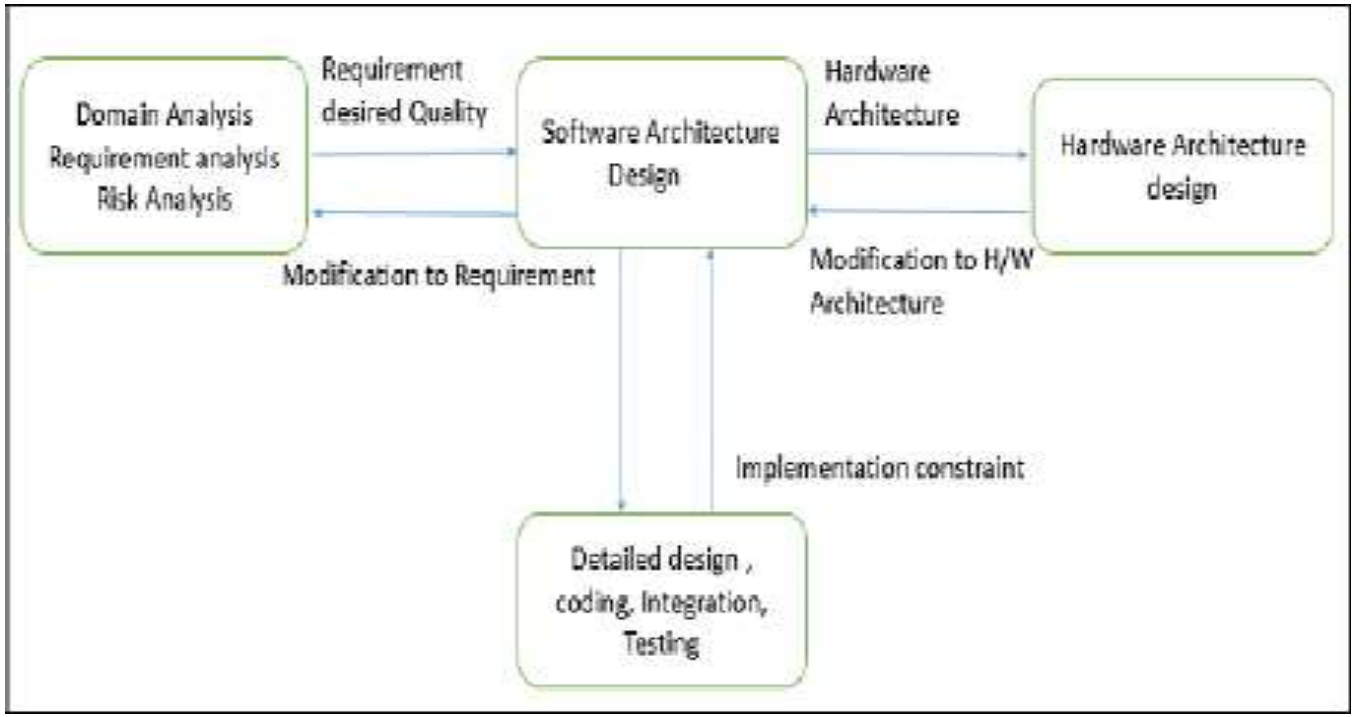
Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.

- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of –
  - Selection of structural elements and their interfaces by which the system is composed.
  - Behavior as specified in collaborations among those elements.
  - Composition of these structural and behavioral elements into large subsystem.
  - Architectural decisions align with business objectives.
  - Architectural styles guide the organization.

Software Design

- Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows –
- To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.
- Act as a blueprint during the development process.
- Guide the implementation tasks, including detailed design, coding, integration, and testing.
- It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.


Domain Analysis
Requirement analysis
Risk Analysis
Requirement
desired Quality
Modification to Requirement
Software Architecture
Design
Hardware
Architecture
Modification to H/W
Architecture
Hardware Architecture
design
Implementation constraint
Detailed design ,
coding, Integration,
Testing

Goals of Architecture

- The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

Some of the other goals are as follows –

- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization's market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

Limitations

Software architecture is still an emerging discipline within software engineering. It has the following limitations –

- Lack of tools and standardized ways to represent architecture.
- Lack of analysis methods to predict whether architecture will result in an implementation that meets the requirements.
- Lack of awareness of the importance of architectural design to software development.
- Lack of understanding of the role of software architect and poor communication among stakeholders.
- Lack of understanding of the design process, design experience and evaluation of design.

Quality Attributes

- Quality is a measure of excellence or the state of being free from deficiencies or defects. Quality attributes are the system properties that are separate from the functionality of the system.
- Implementing quality attributes makes it easier to differentiate a good system from a bad one. Attributes are overall factors that affect runtime behavior, system design, and user experience.

They can be classified as –

Static Quality Attributes

- Reflect the structure of a system and organization, directly related to architecture, design, and source code. They are invisible to end-user, but affect the development and maintenance cost, e.g.: modularity, testability, maintainability, etc.

Dynamic Quality Attributes

- Reflect the behavior of the system during its execution. They are directly related to system's architecture, design, source code, configuration, deployment parameters, environment, and platform.
- They are visible to the end-user and exist at runtime, e.g. throughput, robustness, scalability, etc.

Quality Scenarios

- Quality scenarios specify how to prevent a fault from becoming a failure. They can be divided into six parts based on their attribute specifications –
- **Source** – An internal or external entity such as people, hardware, software, or physical infrastructure that generate the stimulus.
- **Stimulus** – A condition that needs to be considered when it arrives on a system.
- **Environment** – The stimulus occurs within certain conditions.
- **Artifact** – A whole system or some part of it such as processors, communication channels, persistent storage, processes etc.
- **Response** – An activity undertaken after the arrival of stimulus such as detect faults, recover from fault, disable event source etc.
- **Response measure** – Should measure the occurred responses so that the requirements can be tested.

Architectural Style

- The **architectural style**, also called as **architectural pattern**, is a set of principles which shapes an application. It defines an abstract framework for a family of system in terms of the pattern of structural organization.

The architectural style is responsible to –

- Provide a lexicon of components and connectors with rules on how they can be combined.
- Improve partitioning and allow the reuse of design by giving solutions to frequently occurring problems.
- Describe a particular way to configure a collection of components (a module with well-defined interfaces, reusable, and replaceable) and connectors (communication link between modules).

- The software that is built for computer-based systems exhibit one of many architectural styles. Each style describes a system category that encompasses –
- A set of component types which perform a required function by the system.
- A set of connectors (subroutine call, remote procedure call, data stream, and socket) that enable communication, coordination, and cooperation among different components.
- Semantic constraints which define how components can be integrated to form the system.
- A topological layout of the components indicating their runtime interrelationships.

Types of Architecture

- There are four types of architecture from the viewpoint of an enterprise and collectively, these architectures are referred to as **enterprise architecture**.
- **Business architecture** – Defines the strategy of business, governance, organization, and key business processes within an enterprise and focuses on the analysis and design of business processes.
- **Application (software) architecture** – Serves as the blueprint for individual application systems, their interactions, and their relationships to the business processes of the organization.

- **Information architecture** – Defines the logical and physical data assets and data management resources.
- **Information technology (IT) architecture** – Defines the hardware and software building blocks that make up the overall information system of the organization.

Architecture Design Process

- The architecture design process focuses on the decomposition of a system into different components and their interactions to satisfy functional and nonfunctional requirements. The key inputs to software architecture design are –
- The requirements produced by the analysis tasks.

- The hardware architecture (the software architect in turn provides requirements to the system architect, who configures the hardware architecture).
- The result or output of the architecture design process is an **architectural description**. The basic architecture design process is composed of the following steps –

Understand the Problem

- This is the most crucial step because it affects the quality of the design that follows.
- Without a clear understanding of the problem, it is not possible to create an effective solution.

- Many software projects and products are considered failures because they did not actually solve a valid business problem or have a recognizable return on investment (ROI).

Identify Design Elements and their Relationships

- In this phase, build a baseline for defining the boundaries and context of the system.
- Decomposition of the system into its main components based on functional requirements. The decomposition can be modeled using a design structure matrix (DSM), which shows the dependencies between design elements without specifying the granularity of the elements.
- In this step, the first validation of the architecture is done by describing a number of system instances and this step is referred as functionality based architectural design.

Evaluate the Architecture Design

- Each quality attribute is given an estimate so in order to gather qualitative measures or quantitative data, the design is evaluated.
- It involves evaluating the architecture for conformance to architectural quality attributes requirements.
- If all estimated quality attributes are as per the required standard, the architectural design process is finished.
- If not, the third phase of software architecture design is entered: architecture transformation. If the observed quality attribute does not meet its requirements, then a new design must be created.

Transform the Architecture Design

- This step is performed after an evaluation of the architectural design. The architectural design must be changed until it completely satisfies the quality attribute requirements.
- It is concerned with selecting design solutions to improve the quality attributes while preserving the domain functionality.
- A design is transformed by applying design operators, styles, or patterns. For transformation, take the existing design and apply design operator such as decomposition, replication, compression, abstraction, and resource sharing.
- The design is again evaluated and the same process is repeated multiple times if necessary and even performed recursively.
- The transformations (i.e. quality attribute optimizing solutions) generally improve one or some quality attributes while they affect others negatively

Key Architecture Principles

Following are the key principles to be considered while designing an architecture –

- Build to Change Instead of Building to Last
- Consider how the application may need to change over time to address new requirements and challenges, and build in the flexibility to support this.
- Reduce Risk and Model to Analyze
- Use design tools, visualizations, modeling systems such as UML to capture requirements and design decisions. The impacts can also be analyzed. Do not formalize the model to the extent that it suppresses the capability to iterate and adapt the design easily.
- Use Models and Visualizations as a Communication and Collaboration Tool
- Efficient communication of the design, the decisions, and ongoing changes to the design is critical to good architecture.
- Identify and understand key engineering decisions and areas where mistakes are most often made.

- Use an Incremental and Iterative Approach
- Start with baseline architecture and Iteratively add details to the design over multiple passes to get the big or right picture and then focus on the details.

Key Design Principles

Following are the design principles to be considered for minimizing cost, maintenance requirements, and maximizing extendibility, usability of architecture

Separation of Concerns

- Divide the components of system into specific features so that there is no overlapping among the components functionality. This will provide high cohesion and low coupling. This approach avoids the interdependency among components of system which helps in maintaining the system easy.

Single Responsibility Principle

- Each and every module of a system should have one specific responsibility, which helps the user to clearly understand the system. It should also help with integration of the component with other components.

Principle of Least Knowledge

- Any component or object should not have the knowledge about internal details of other components. This approach avoids interdependency and helps maintainability.

Minimize Large Design Upfront

- Minimize large design upfront if the requirements of an application are unclear. If there is a possibility of modifying requirements, then avoid making a large design for whole system.

Do not Repeat the Functionality

- Do not repeat functionality specifies that functionality of components should not to be repeated and hence a piece of code should be implemented in one component only. Duplication of functionality within an application can make it difficult to implement changes, decrease clarity, and introduce potential inconsistencies.

Prefer Composition over Inheritance while Reusing the Functionality

- Inheritance creates dependency between children and parent classes and hence it blocks the free use of the child classes. In contrast, the composition provides a great level of freedom and reduces the inheritance hierarchies.

Identify Components and Group them in Logical Layers

- Identity components and the area of concern that are needed in system to satisfy the requirements. Then group these related components in a logical layer, which will help the user to understand the structure of the system at a high level. Avoid mixing components of different type of concerns in same layer.

Define the Communication Protocol between Layers

- Understand how components will communicate with each other which requires a complete knowledge of deployment scenarios and the production environment.

Define Data Format for a Layer

- Various components will interact with each other through data format. Do not mix the data formats so that applications are easy to implement, extend, and maintain.

System Service Components should be Abstract

- Code related to security, communications, or system services like logging, profiling, and configuration should be abstracted in the separate components. Do not mix this code with business logic, as it is easy to extend design and maintain it.

Design Exceptions and Exception Handling Mechanism

- Defining exceptions in advance, helps the components to manage errors or unwanted situation in an elegant manner. The exception management will be same throughout the system.

Naming Conventions

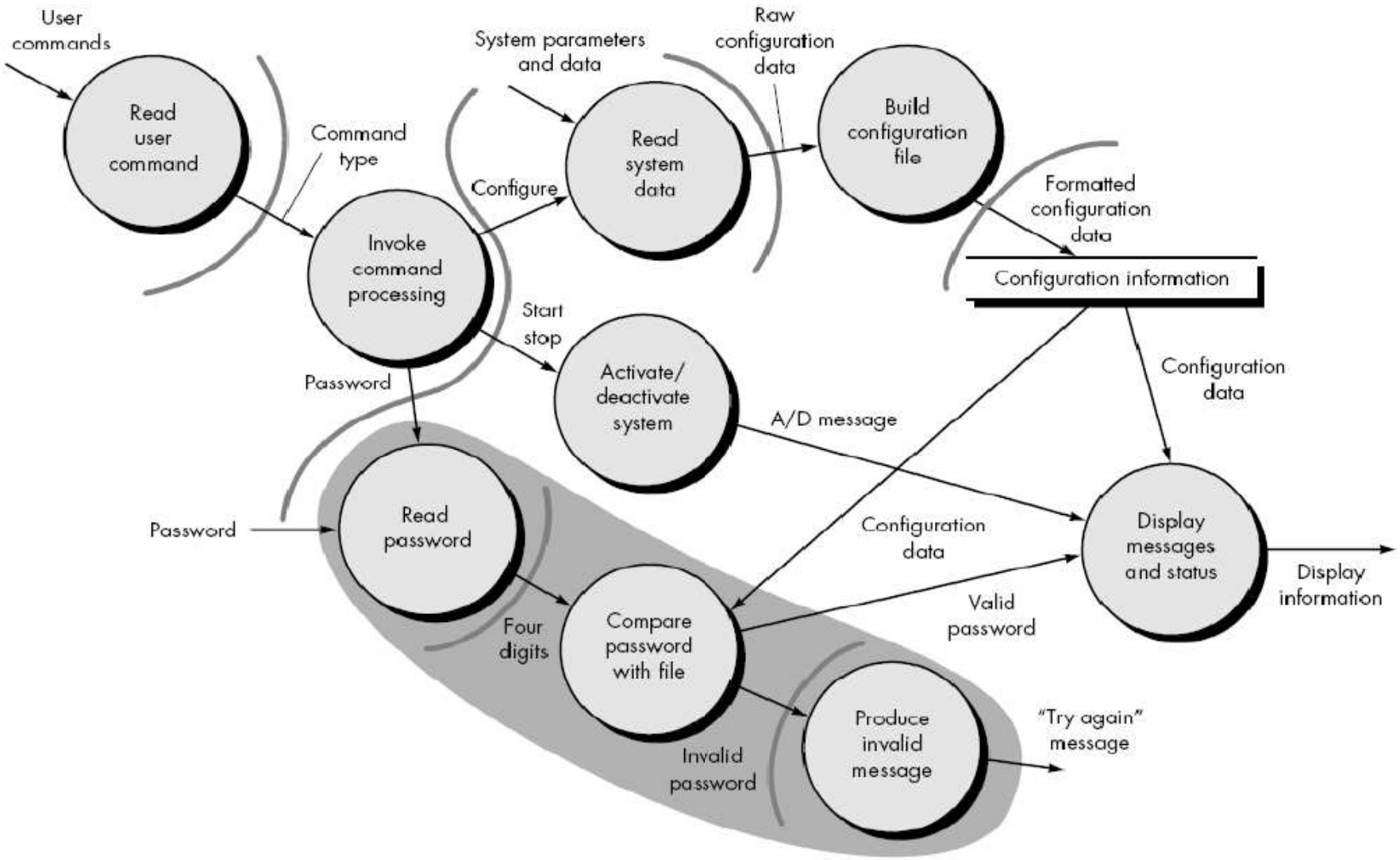
- Naming conventions should be defined in advance. They provide a consistent model that helps the users to understand the system easily. It is easier for team members to validate code written by others, and hence will increase the maintainability.

Transaction Mapping

In many software applications, a single data item triggers one or a number of information flows that effect a function implied by the triggering data item. The data item, called a transaction, and its corresponding flow characteristics . In this section we consider design steps used to treat transaction flow.

An Example

Transaction mapping will be illustrated by considering the user interaction subsystem of the SafeHome software.


User commands
Read user command
Command type
Invoke command processing
Configure
System parameters and data
Read system data
Raw configuration data
Build configuration file
Formatted configuration data
Configuration information
Start stop
Activate/ deactivate system
Password
A/D message
Configuration data
Password
Read password
Four digits
Compare password with file
Configuration data
Valid password
Display messages and status
Display information
Invalid password
Produce invalid message
"Try again" message

- As shown in the figure, user commands flows into the system and results in additional information flow along one of three action paths. A single data item, command type, causes the data flow to fan outward from a hub. Therefore, the overall data flow characteristic is transaction oriented.

It should be noted that information flow along two of the three action paths accommodate additional incoming flow (e.g., system parameters and data are input on the "configure" action path). Each action path flows into a single transform, display messages and status.

Design Steps

- The design steps for transaction mapping are similar and in some cases identical to steps for transform mapping . A major difference lies in the mapping of DFD to software structure.

Step 1. Review the fundamental system model.
Step 2. Review and refine data flow diagrams for the software.
Step 3. Determine whether the DFD has transform or transaction flow characteristics.

Steps 1, 2, and 3 are identical to corresponding steps in transform mapping.

Step 4.

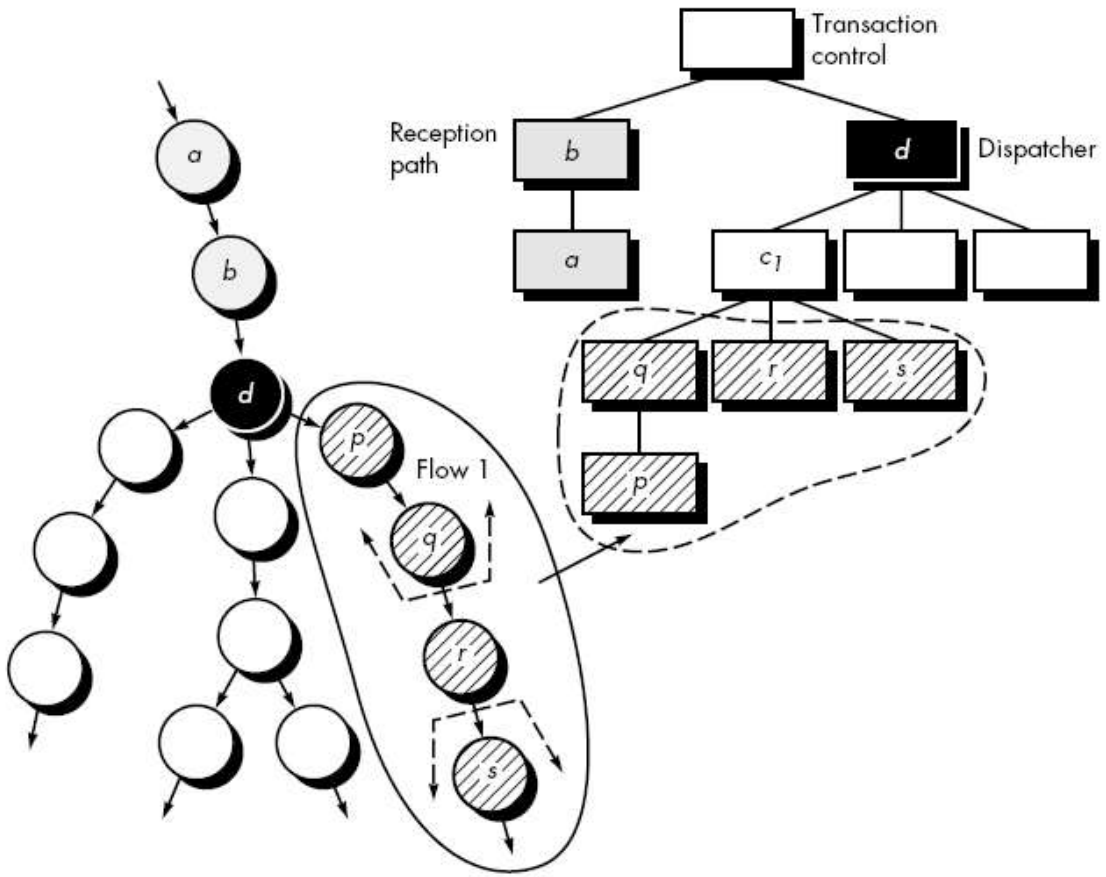
Identify the transaction center and the flow characteristics along each of the action paths. The location of the transaction center can be immediately discerned from the DFD.

The incoming path (i.e., the flow path along which a transaction is received) and all action paths must also be isolated. Boundaries that define a reception path and action paths are also shown in the figure.

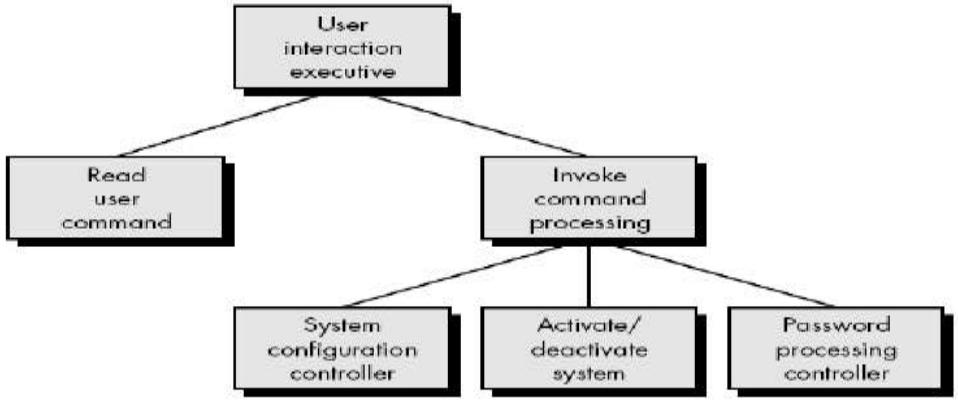
Each action path must be evaluated for its individual flow characteristic.

For example, the "password" path has transform characteristics. Incoming, transform, and outgoing flow are indicated with boundaries.

- **Step 5.** Map the DFD in a program structure amenable to transaction processing.
- Transaction flow is mapped into an architecture that contains an incoming branch and a dispatch branch.
- Starting at the transaction center, bubbles along the incoming path are mapped into modules.
- The structure of the dispatch branch contains a dispatcher module that controls all subordinate action modules.
- Each action flow path of the DFD is mapped to a structure that corresponds to its specific flow characteristics. This process is illustrated schematically in figure below.


Transaction control
Reception path
b
d
Dispatcher
a
c_1
q
r
s
p
a
b
d
p
Flow 1
q
r
s

- Considering the user interaction subsystem data flow, first-level factoring for step 5 is shown in below figure.



The bubbles read user command and activate/deactivate system map directly into the architecture without the need for intermediate control modules. The transaction center, invoke command processing, maps directly into a dispatcher module of the same name. Controllers for system configuration and password processing are created as illustrated in figure.

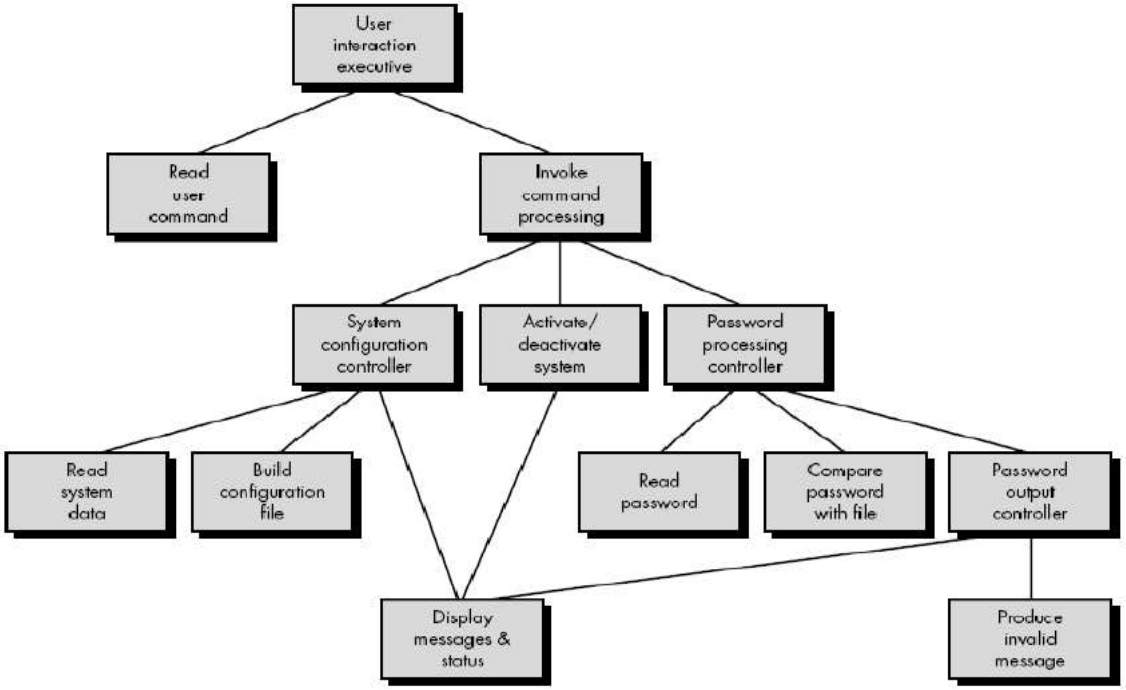
- Step 6. Factor and refine the transaction structure and the structure of each action path.
- Each action path of the data flow diagram has its own information flow characteristics.

As an example, consider the password processing information flow shown (inside shaded area) in figure

The flow exhibits classic transform characteristics. A password is input (incoming flow) and transmitted to a transform center where it is compared against stored passwords.

An alarm and warning message (outgoing flow) are produced (if a match is not obtained).

The "configure" path is drawn similarly using the transform mapping. The resultant software architecture is shown in the figure below



Step 7. Refine the first-iteration architecture using design heuristics for improved software quality. This step for transaction mapping is identical to the corresponding step for transform mapping. In both design approaches, criteria such as module independence, practicality (efficacy of implementation and test), and maintainability must be carefully considered as structural modifications are proposed.

Transform Mapping

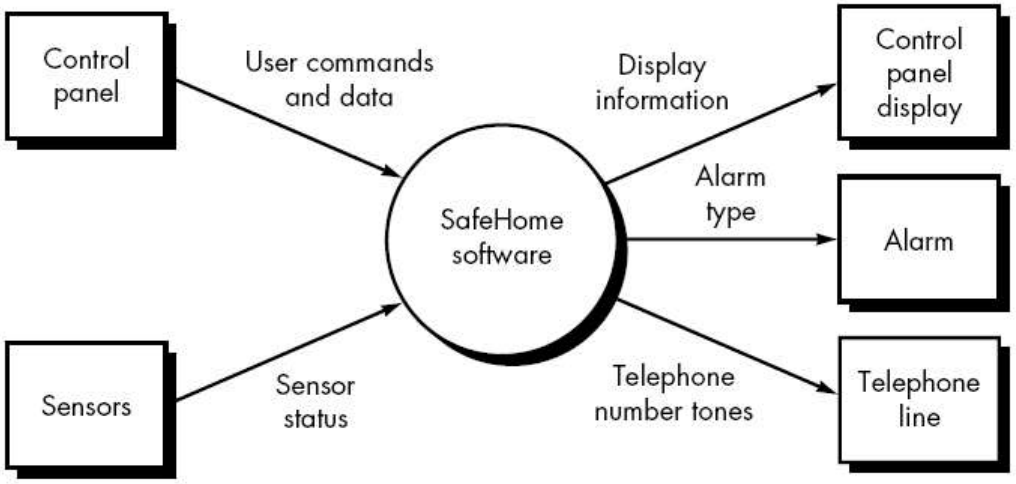
Transform mapping is a set of design steps that allows a DFD with transform flow characteristics to be mapped into a specific architectural style.

example system—a portion of the SafeHome security software.

An Example

This product monitors the real world and reacts to changes that it encounters.

It also interacts with a user through a series of typed inputs and alphanumeric displays. The level 0 data flow diagram for SafeHome, is shown in figure



During requirements analysis, more detailed flow models would be created for SafeHome. In addition, control and process specifications, a data dictionary, and various behavioral models would also be created.

Design Steps

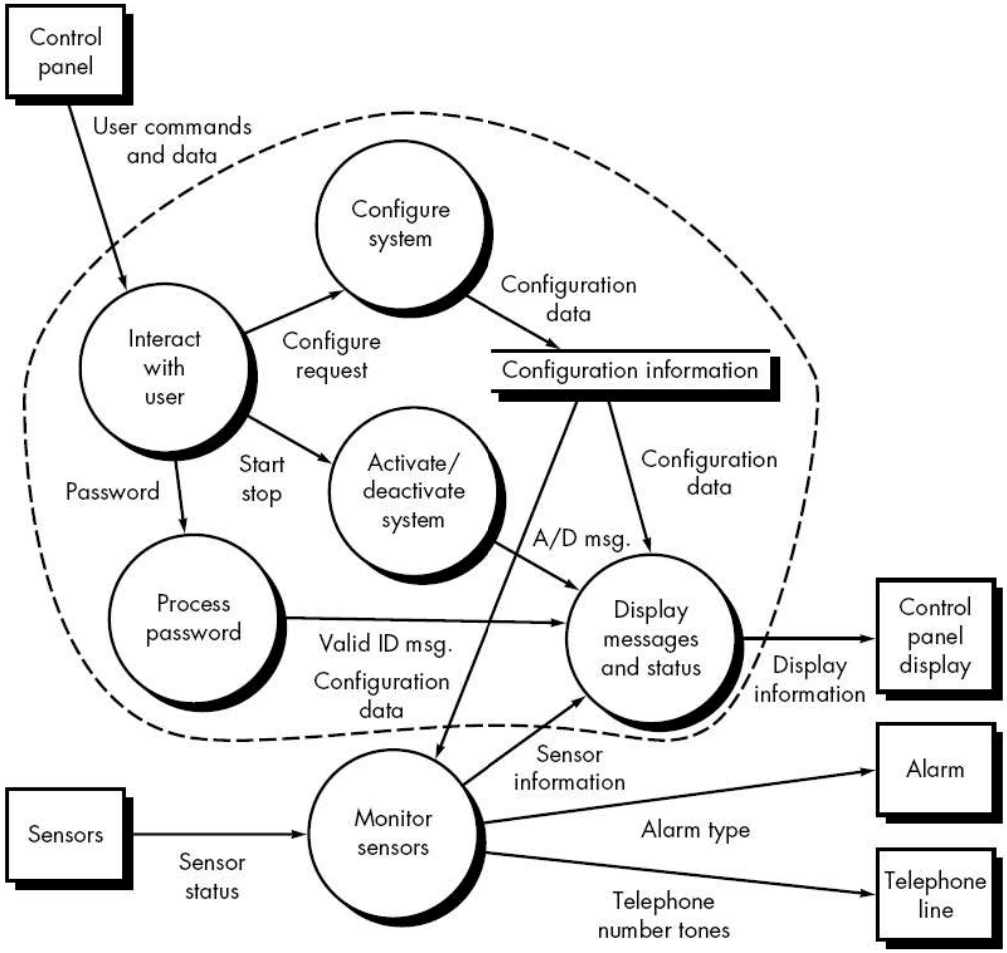
The preceding example will be used to illustrate each step in transform mapping. The steps begin with a re-evaluation of work done during requirements analysis and then move to the design of the software architecture.

Step 1. Review the fundamental system model.

The fundamental system model encompasses the level 0 DFD and supporting information.

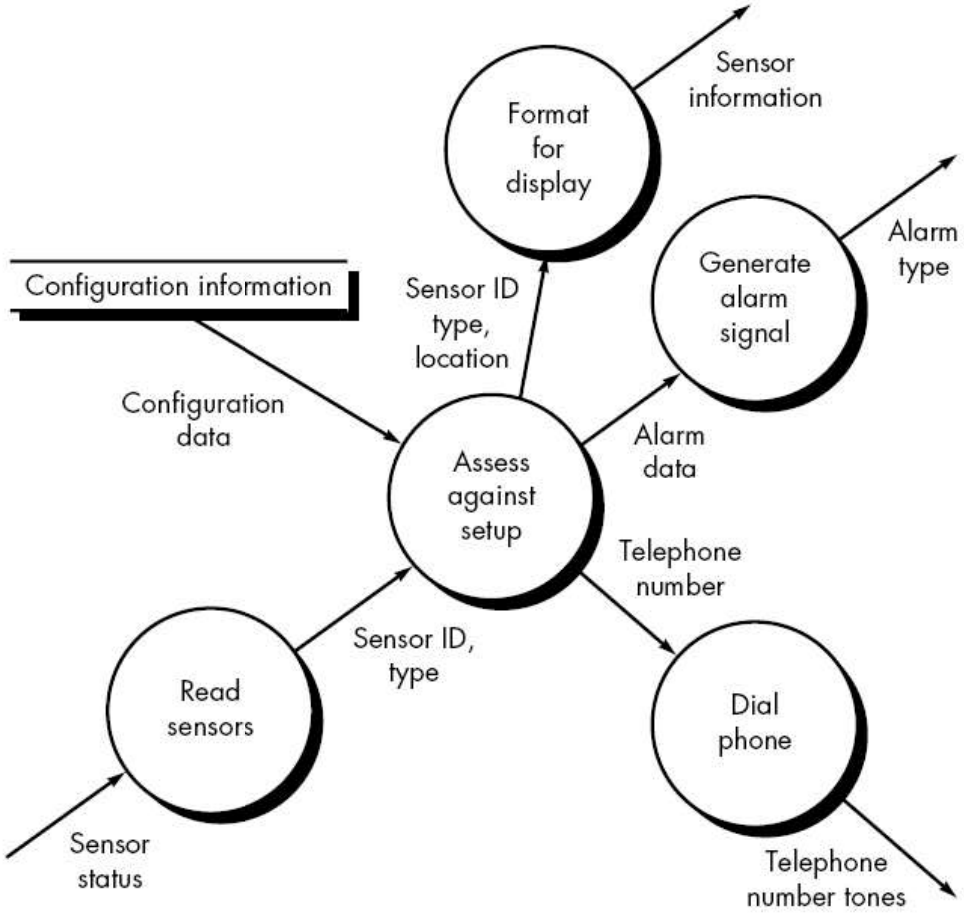
The design step begins with an evaluation of both the System Specification and the Software Requirements Specification.

Both documents describe information flow and structure at the software interface. Figure 1 and 2 depict level 0 and level 1 data flow for the SafeHome software.


Control panel
User commands and data
Configure system
Configuration data
Interact with user
Configure request
Configuration information
Start stop
Activate/ deactivate system
Configuration data
Password
A/D msg.
Process password
Display messages and status
Control panel display
Valid ID msg.
Configuration data
Display information
Sensor information
Alarm
Sensors
Monitor sensors
Alarm type
Sensor status
Telephone line
Telephone number tones

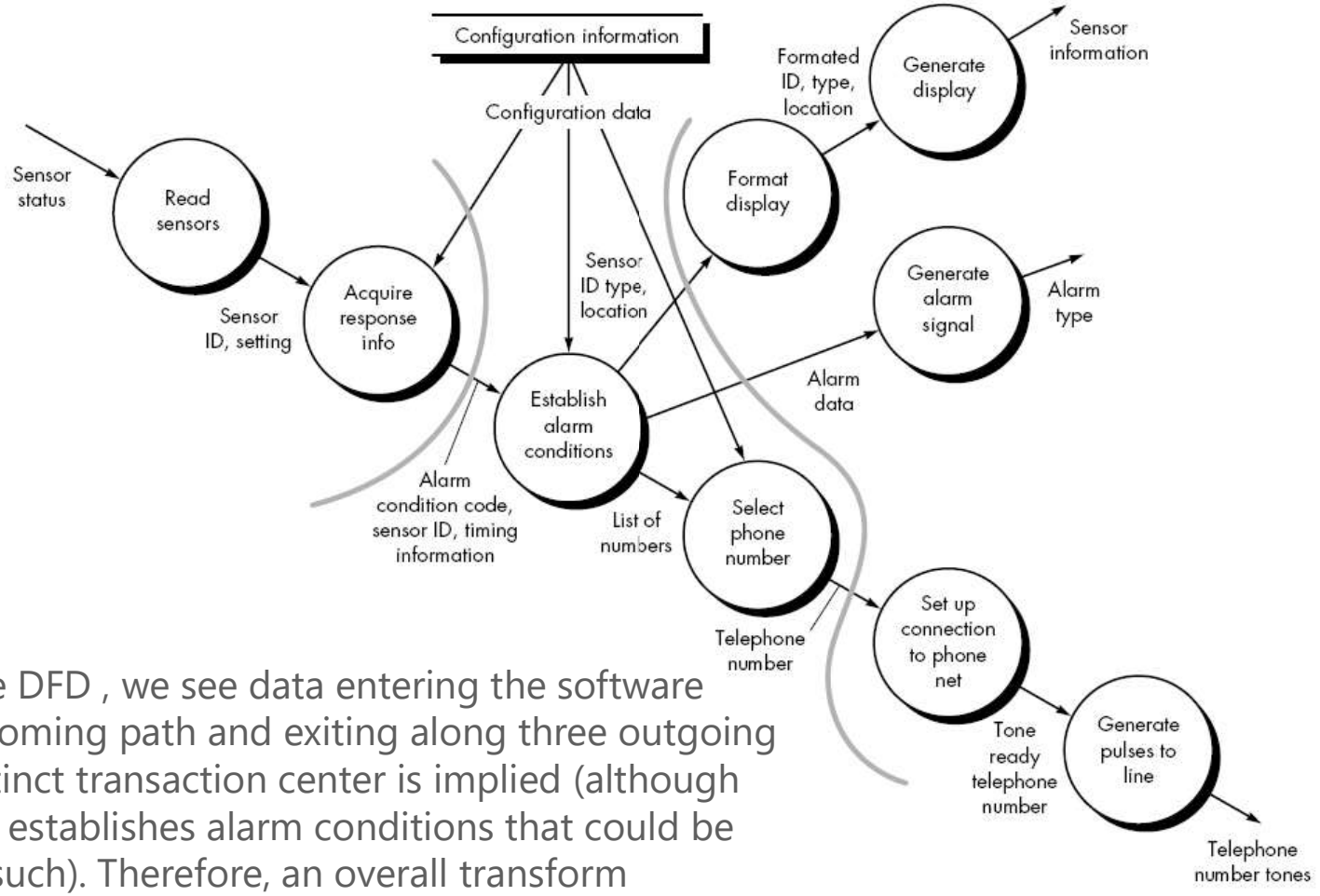
Step 2. Review and refine data flow diagrams for the software.

- Information obtained from analysis models contained in the Software Requirements Specification is refined to produce greater detail.
- For example, the level 2 DFD for monitor sensors is examined, and a level 3 data flow diagram is derived .
- At level 3, each transform in the data flow diagram exhibits relatively high cohesion.
- That is, the process implied by a transform performs a single, distinct function that can be implemented in the SafeHome software.


Format for display
Sensor information
Configuration information
Sensor ID type, location
Generate alarm signal
Alarm type
Configuration data
Assess against setup
Alarm data
Telephone number
Read sensors
Sensor ID, type
Dial phone
Sensor status
Telephone number tones

Step 3. Determine whether the DFD has transform or transaction flow characteristics.

- In general, information flow within a system can always be represented as transform.
- In this step, the designer selects global flow characteristics based on the prevailing nature of the DFD.
- In addition, local regions of transform or transaction flow are isolated.
- These subflows can be used to refine program architecture derived from a global characteristic described previously.
- For now, we focus our attention only on the monitor sensors subsystem data flow depicted in figure.



Evaluating the DFD , we see data entering the software along one incoming path and exiting along three outgoing paths. No distinct transaction center is implied (although the transform establishes alarm conditions that could be perceived as such). Therefore, an overall transform characteristic will be assumed for information flow.

Step 4. Isolate the transform center by specifying incoming and outgoing flow boundaries.

- In the preceding section incoming flow was described as a path in which information is converted from external to internal form;
- outgoing flow converts from internal to external form. Incoming and outgoing flow boundaries are open to interpretation.
- That is, different designers may select slightly different points in the flow as boundary locations.

- **Step 5. Perform "first-level factoring." Program structure represents a top-down distribution of control.**
- Factoring results in a program structure in which top-level modules perform decision making and low-level modules perform most input, computation, and output work. Middle-level modules perform some control and do moderate amounts of work.

•

- **Step 6. Perform "second-level factoring." Second-level factoring is accomplished by mapping individual transforms (bubbles) of a DFD into appropriate modules within the architecture.**
- Beginning at the transform center boundary and moving outward along incoming and then outgoing paths, transforms are mapped into subordinate levels of the software structure.

- **Step 7. Refine the first-iteration architecture using design heuristics for improved software quality.**
- A first-iteration architecture can always be refined by applying concepts of module independence . Modules are exploded or imploded to produce sensible factoring, good cohesion, minimal coupling, and most important, a structure that can be implemented without difficulty, tested without confusion, and maintained without grief.

Refactoring of designs

Refactoring is the process of restructuring code, while not changing its original functionality. The goal of refactoring is to improve internal code by making many small changes without altering the code's external behavior.

software developers refactor code to improve the design, structure and implementation of software. Refactoring improves code readability and reduces complexities.

These changes preserve the software's original behavior and do not modify its behavior.

Purpose of refactoring

Refactoring improves code by making it:

- More efficient by addressing dependencies and complexities.
- More maintainable or reusable by increasing efficiency and readability.
- Cleaner so it is easier to read and understand.
- Easier for software developers to find and fix bugs or vulnerabilities in the code.

- Code modification is done without changing any functions of the program itself. Many basic editing environments support simple refactorings like renaming a function or variable across an entire code base.
- Refactoring can be performed after a product has been deployed, before adding updates and new features to existing code, or as a part of day-to-day programming.
- A better time to perform refactoring, is before adding updates or new features to existing code.

Benefits of refactoring

- Makes the code easier to understand and read because the goal is to simplify code and reduce complexities.
- Improves maintainability and makes it easier to spot bugs or make further changes.
- Encourages a more in-depth understanding of code. Developers have to think further about how their code will mix with code already in the code base.
- Focus remains only on functionality. Not changing the code's original functionality ensures the original project does not lose scope.

challenges of refactoring

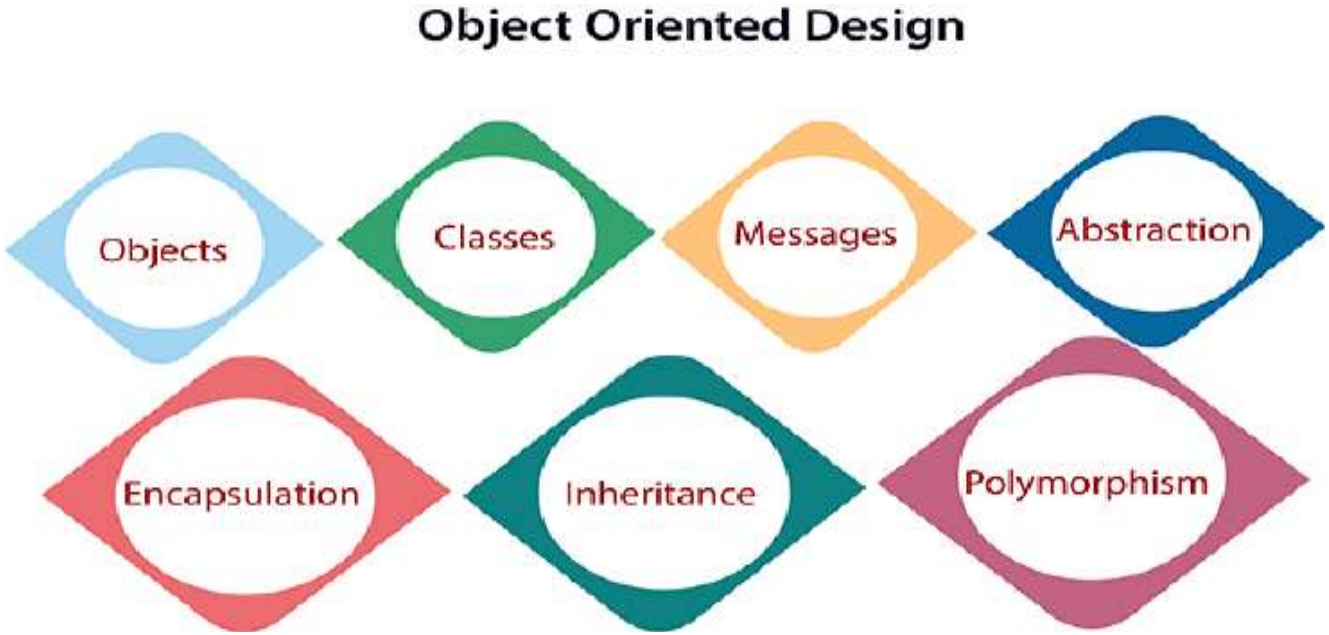
- The process will take extra time if a development team is in a rush and refactoring is not planned for.
- Without clear objectives, refactoring can lead to delays and extra work.
- Refactoring cannot address software flaws by itself, as it is made to clean code and make it less complex.

Code refactoring best practices

- **Plan for refactoring.** It may be difficult to make time for the time-consuming practice otherwise.
- **Refactor first.** Developers should do this before adding updates or new features to existing code to reduce technical debt.
- **Refactor in small steps.** This gives developers feedback early in the process so they can find possible bugs, as well as include business requests.
- **Set clear objectives.** Developers should determine the project scope and goals early in the code refactoring process. This helps to avoid delays and extra work, as refactoring is meant to be a form of housekeeping, not an opportunity to changes functions or features.
- **Test often.** This helps to ensure refactored changes do not introduce new bugs.
- **Automate wherever possible.** Automation tools make refactoring easier and faster, thus, improving efficiency.
- **Fix software defects separately.** Refactoring is not meant to address software flaws. Troubleshooting and debugging should be done separately.
- **Understand the code.** Review the code to understand its processes, methods, objects, variables and other elements.
- **Refactor, patch and update regularly.** Refactoring generates the most return on investment when it can address a significant issue without taking too much time and effort.
- **Focus on code deduplication.** Duplication adds complexities to code, expanding the software's footprint and wasting system resources.

Object-Oriented Design

- In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data.



1. **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
2. **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.
3. **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.

3.Abstraction In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.

5.Encapsulation: Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.

6.Inheritance: OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses.This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.

7. Polymorphism: OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

User Interface Design

- The visual part of a computer application or operating system through which a client interacts with a computer or software. It determines how commands are given to the computer or the program and how data is displayed on the screen.

Types of User Interface

- There are two main types of User Interface:
- Text-Based User Interface or Command Line Interface
- Graphical User Interface (GUI)

Text-Based User Interface: This method relies primarily on the keyboard. A typical example of this is UNIX.

Advantages

- Many and easier to customizations options.
- Typically capable of more important tasks.
- Disadvantages
- Relies heavily on recall rather than recognition.
- Navigation is often more difficult.

Graphical User Interface (GUI): GUI relies much more heavily on the mouse. A typical example of this type of interface is any versions of the Windows operating systems.

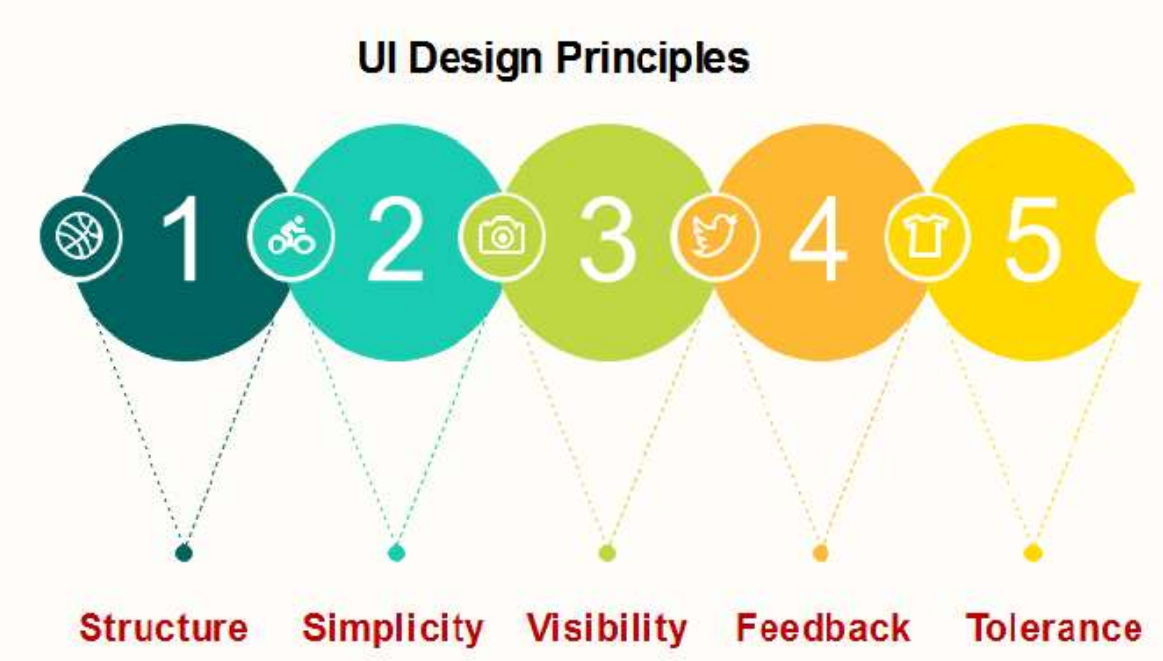
- GUI Characteristics

Characteristics	Descriptions
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons different types of information. On some systems, icons represent files. On other icons describes processes.
Menus	Commands are selected from a menu rather than typed in a command language.
Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interests in a window.
Graphics	Graphics elements can be mixed with text or the same display.

Advantages

- Less expert knowledge is required to use it.
- Easier to Navigate and can look through folders quickly in a guess and check manner.
- The user may switch quickly from one task to another and can interact with several different applications.
- Disadvantages
- Typically decreased options.
- Usually less customizable. Not easy to use one button for tons of different variations.

UI Design Principles



- **Structure:** Design should organize the user interface purposefully, in the meaningful and usual based on precise, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.
- **Simplicity:** The design should make the simple, common task easy, communicating clearly and directly in the user's language, and providing good shortcuts that are meaningfully related to longer procedures.
- **Visibility:** The design should make all required options and materials for a given function visible without distracting the user with extraneous or redundant data.
- **Feedback:** The design should keep users informed of actions or interpretation, changes of state or condition, and bugs or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.
- **Tolerance:** The design should be flexible and tolerant, decreasing the cost of errors and misuse by allowing undoing and redoing while also preventing bugs wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

BCSE301L SOFTWARE ENGINEERING

Module:5 Validation And Verification

Strategic Approach to Software Testing, Testing Fundamentals Test Plan, Test Design, Test Execution, Reviews, Inspection and Auditing – Regression Testing – Mutation Testing - Object oriented testing - Testing Web based System - Mobile App testing – Mobile test Automation and tools – DevOps Testing – Cloud and Big Data Testing

Test plan(procedures)

- A test plan is a detailed document which describes software testing areas and activities. It outlines the test strategy, objectives, test schedule, required resources (human resources, software, and hardware), test estimation and test deliverables.
- The test plan is a template for conducting software testing activities as a defined process that is fully monitored and controlled by the testing manager. The test plan is prepared by the Test Lead (60%), Test Manager(20%), and by the test engineer(20%).

Types of Test Plan

There are three types of the test plan

- Master Test Plan
- Phase Test Plan
- Testing Type Specific Test Plans

Master Test Plan

- Master Test Plan is a type of test plan that has multiple levels of testing. It includes a complete test strategy.

Phase Test Plan

- A phase test plan is a type of test plan that addresses any one phase of the testing strategy. For example, a list of tools, a list of test cases, etc.

Specific Test Plans

- Specific test plan designed for major types of testing like security testing, load testing, performance testing, etc. In other words, a specific test plan designed for non-functional testing.

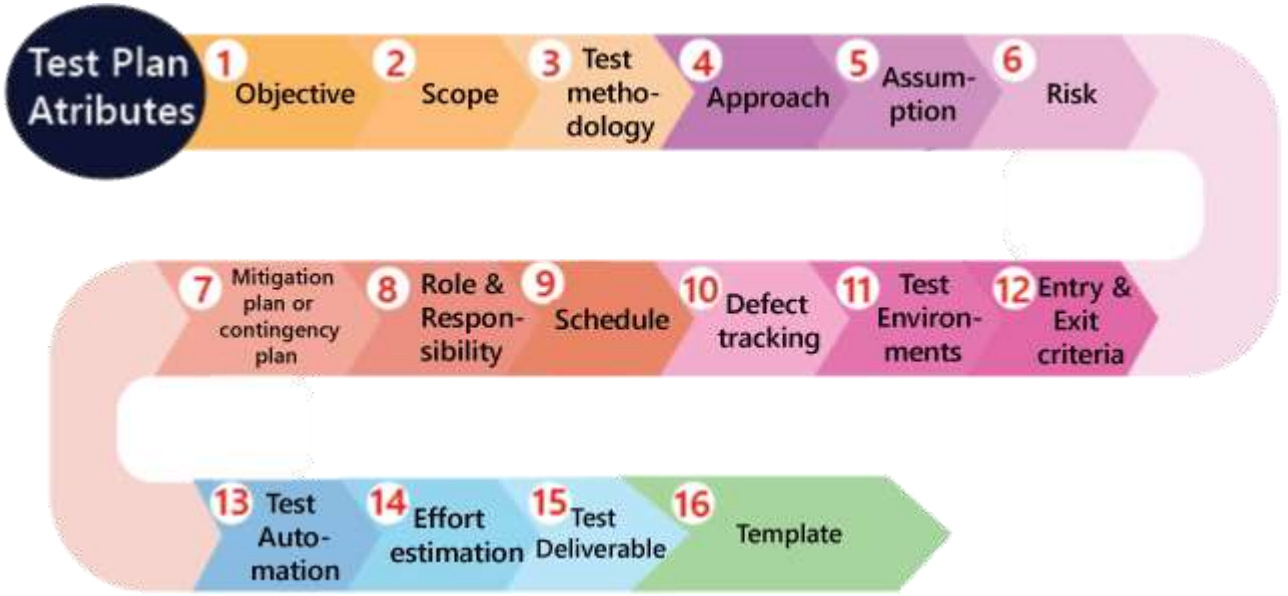
How to write a Test Plan

Making a test plan is the most crucial task of the test management process. seven steps to prepare a test plan.

- First, analyze product structure and architecture.
- Now design the test strategy.
- Define all the test objectives.
- Define the testing area.
- Define all the useable resources.
- Schedule all activities in an appropriate manner.
- Determine all the Test Deliverables.

Test plan components or attributes

The test plan consists of various parts, which help us to derive the entire testing activity.



Test Plan Template

Below find important constituents of a test plan-

- 1 Introduction
- 1.1 Scope
  - 1.1.1 In Scope
  - 1.1.2 Out of Scope
- 1.2 Quality Objective
- 1.3 Roles and Responsibilities
- 2 Test Methodology
- 2.1 Overview
- 2.2 Test Levels
- 2.3 Bug Triage
- 2.4 Suspension Criteria and Resumption Requirements
- 2.5 Test Completeness
- 3 Test Deliverables
- 4 Resource & Environment Needs
- 4.1 Testing Tools
- 4.2 Test Environment

- **Objectives:** It consists of information about modules, features, test data etc., which indicate the aim of the application means the application behavior, goal, etc.
- **Scope:** It contains information that needs to be tested with respective of an application. The Scope can be further divided into two parts:

In scope

Out scope

- **In scope:** These are the modules that need to be tested rigorously (in-detail).
- **Out scope:** These are the modules, which need not be tested rigorously.
- **For example**, Suppose we have a Gmail application to test, where **features to be tested** such as **Compose mail, Sent Items, Inbox, Drafts** and the **features which not be tested** such as **Help**, and so on which means that in the planning stage, we will decide that which functionality has to be checked or not based on the time limit given in the product.

Quality Objective

- Here make a mention of the overall objective that you plan to achieve with your manual testing and automation testing.

Some objectives of your testing project could be

- Ensure the Application Under Test (AUT)conforms to functional and non-functional requirements
- Ensure the AUT meets the quality specifications defined by the client
- Bugs/issues are identified and fixed before go live

Roles and Responsibilities

Detail description of the Roles and responsibilities of different team members like

- QA Analyst
- Test Manager
- Configuration Manager
- Developers
- Installation Team

2) Test Methodology

2.1) Overview

Mention the reason of adopting a particular test methodology for the project. The test methodology selected for the project could be

- Waterfall
- Iterative
- Agile
- Extreme Programming

2.2) Test Levels

Test Levels define the Types of Testing to be executed on the Application Under Test (AUT). The Testing Levels primarily depends on the scope of the project, time and budget constraints.

2.3) Bug Triage

The goal of the triage is to

- To define the type of resolution for each bug
- To prioritize bugs and determine a schedule for all “To Be Fixed Bugs’.

2.4) Suspension Criteria and Resumption Requirements

- Suspension criteria define the criteria to be used to suspend all or part of the testing procedure while Resumption criteria determine when testing can resume after it has been suspended.

2.5) Test Completeness

Here you define the criteria's that will deem your testing complete.

- For instance, a few criteria to check Test Completeness would be 100% test coverage
- All Manual & Automated Test cases executed
- All open bugs are fixed or will be fixed in next release

3) Test Deliverables

- Here mention all the Test Artifacts that will be delivered during different phases of the testing lifecycle.

Here are the simple deliverables

- Test Plan
- Test Cases
- Requirement Traceability Matrix
- Bug Reports
- Test Strategy
- Test Metrics
- Customer Sign Off

Requirement no	Module name	High level requirement	Low level requirement	Test case name

	A	B	C	D	E
1	RTM Template				
2	**Requirement number**	**Module number**	**High level requirement**	**Low level requirement**	**Test case name**
3	2	Loan	**2.1 Personal loan**	2.1.1--> personal loan for private employee	beta-2.0-personal loan
4				2.1.2--> personal loan for government employee	
5				2.1.3--> personal loan for jobless people	
6			**2.2 Car loan**	2.2.1--> car loan for private employee	
7				——	
8			**2.3 Home loan**	——	
9				——	
10				——	
11					

4) Resource & Environment Needs

4.1) Testing Tools

Make a list of Tools like

- Requirements Tracking Tool
- Bug Tracking Tool
- Automation Tools

4.2) Test Environment

- It mentions the minimum **hardware** requirements that will be used to test the Application.

Following **software's** are required in addition to client-specific software.

- Windows 8 and above
- Office 2013 and above
- MS Exchange, etc.

Sample Test Plan Document Banking Web Application Example

1 Introduction

- The Test Plan is designed to prescribe the scope, approach, resources, and schedule of all testing activities of the project.
- The plan identify the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.
- **1.1 Scope**
- **1.1.1 In Scope**
- All the feature of the Banking web application which were defined in software requirement and the specifications are to be tested

Module Name	Applicable Roles	Description
Balance Enquiry	Manager Customer	**Customer**: A customer can have multiple bank accounts. He can view balance of his accounts only **Manager**: A manager can view balance of all the customers who come under his supervision
Fund Transfer	Manager Customer	**Customer:** A customer can have transfer funds from his “own” account to any destination account. **Manager**: A manager can transfer funds from any source bank account to destination account
Mini Statement	Manager Customer	A Mini statement will show last 5 transactions of an account **Customer:** A customer can see mini-statement of only his “own” accounts **Manager:** A manager can see mini-statement of any account

Customized Statement	Manager Customer	A customized statement allows you to filter and display transactions in an account based on date, transaction value **Customer:** A customer can see Customized- statement of only his “own” accounts **Manager**: A manager can see Customized -statement of any account
Change Password	Manager Customer	**Customer:** A customer can change password of only his account. **Manager**: A manager can change password of only his account. He cannot change passwords of his customers
New Customer	Manager	**Manager**: A manager can add a new customer.
	Manager	**Manager:** A manager can edit details like address, email, telephone of a customer.

New Account	Manager	A customer can have multiple saving accounts (one in his name, other in a joint name etc). He can have multiple current accounts for different companies he owns. Or he can have a multiple current and saving accounts. **Manager:** A manager can add a new account for an existing customer.
Edit Account	Manager	**Manager:** A manager can add a edit account details for an existing account
Delete Account	Manager	**Manager:** A manager can add a delete an account for a customer.
Delete Customer	Manager	A customer can be deleted only if he/she has no active current or saving accounts **Manager:** A manager can delete a customer.
Deposit	Manager	**Manager:** A manager can deposit money into any account. Usually done when cash is deposited at a bank branch.
Withdrawal	Manager	**Manager:** A manager can withdraw money from any account. Usually done when cash is withdrawn at a bank branch.

1.1.2 Out of Scope

These feature are not be tested because they are not included in the software requirement specs

- User Interfaces
- Hardware Interfaces
- Software Interfaces
- Database logical
- Communications Interfaces
- Website Security and Performance

1.2 Quality Objective

- The test objectives are to **verify** the Functionality of bank website, the project should focus on testing the **banking operation** such as Account Management, Withdrawal, and Balance.

• 1.3 Roles and Responsibilities

No.	Member	Tasks
1.	Test Manager	Manage the whole project Define project directions Acquire appropriate resources
2.	Test	Identifying and describing appropriate test techniques/tools/automation architecture Verify and assess the Test Approach Execute the tests, Log results, Report the defects. Outsourced members
3.	Developer in Test	Implement the test cases, test program, test suite etc.
4.	Test Administrator	Builds up and ensures test environment and assets are managed and maintained Support Tester to use the test environment for test execution
5.	SQA members	Take in charge of quality assurance Check to confirm whether the testing process is meeting specified requirements

2 Test Methodology

2.1 Overview

2.2 Test Levels

- In the project, there're 3 types of testing should be conducted.
- **Integration** Testing (Individual software modules are combined and tested as a group)
- **System** Testing: Conducted on a **complete**, **integrated** system to evaluate the system's compliance with its specified requirements
- **API testing:** Test all the APIs create for the software under tested

2.3 Bug Triage

2.4 Suspension Criteria and Resumption Requirements

- If the team members report that there are **40%** of test cases **failed**, suspend testing until the development team fixes all the failed cases.

2.5 Test Completeness

- Specifies the criteria that denote a **successful** completion of a test phase
- **Run** rate is mandatory to be **100%** unless a clear reason is given.
- **Pass** rate is **80%,** achieving the pass rate is **mandatory**

2.6 Project task and estimation and schedule

Task	Members	Estimate effort
Create the test specification	Test Designer	170 man-hour
Perform Test Execution	Tester, Test Administrator	80 man-hour
Test Report	Tester	10 man-hour
Test Delivery		20 man-hour
Total		280 man-hour

Schedule to complete these tasks

3 .Test Deliverables

Test deliverables are provided as below

Before testing phase

- Test plans document.
- Test cases documents
- Test Design specifications.

During the testing

- – Test Tool Simulators.
- – Test Data
- – Test Trace-ability Matrix – Error logs and execution logs.

After the testing cycles is over

- Test Results/reports
- Defect Report
- Installation/ Test procedures guidelines
- Release notes

4 Resource & Environment Needs

4.1 Testing Tools

No.	Resources	Descriptions
1.	Server	Need a Database server which install MySQL server Web server which install Apache Server
2.	Test tool	Develop a Test tool which can auto generate the test result to the predefined form and automated test execution
3.	Network	Setup a LAN Gigabit and 1 internet line with the speed at least 5 Mb/s
4.	Computer	At least 4 computer run Windows 7, Ram 2GB, CPU 3.4GHZ

4.2 Test Environment

- It mentions the minimum hardware and software requirements that will be used to test the Application.
- Following software’s are required in addition to client-specific software.
- Windows 11 and above
- Office 2021 and above
- MS Exchange, etc.

Test Design

- Test design is a process that describes “how” testing should be done. It includes processes for the identifying test cases by enumerating steps of the defined test conditions.
- The test cases may be linked to the test conditions and project objectives directly or indirectly depending upon the methods used for test monitoring, control and traceability.
- The objectives consist of test objectives, strategic objectives and stakeholder definition of success.

When to create test design?

- After the test conditions are defined and sufficient information is available to create the test cases of high or low level, test design for a specified level can be created.
- For lower level testing, test analysis(what is to be tested) and design(test cases) are combined activity. For higher level testing, test analysis is performed first, followed by test design.
- There are some activities that routinely take place when the test is implemented. These activities may also be incorporated into the design process when the tests are created in an iterative manner.

Test Implementation

- Test implementation is the practice of organizing and prioritizing tests. This is carried out by Test Analysts who implement the test designs as real test cases, test processes and test data.
- If you are observing the IEEE 829 standard, you must define these parameters as well :
- Test inputs
- Expected results for each test case
- Steps to be followed for each test process

Some of the checks that could be performed to confirm that the team ready to execute tests include:

- Ensuring that the test environment is in place
- Ensuring every test case is well documented and reviewed
- Putting test environment in a state of readiness
- Checking against explicit and implicit entry criteria for the specified test level
- Describing test environment as well as test data in great detail
- Performing code acceptance check by running it on test environment
- For example, if the tests are to be documented for using again in future for regression testing, the test documents will record step by step description of executing the test.

Disadvantages of early test implementation

- Implementing the tests early may have some disadvantages too.
- For example, if Agile lifecycle has been adopted for product development, the code itself may undergo drastic changes between consecutive iterations. This will render the whole test implementation useless.
- In fact, any iterative development lifecycle will affect the code between iterations, even if it is not as drastic as that in the Agile lifecycle.
- This will make pre-defined tests obsolete or require continuous and resource intensive maintenance.
- Similarly, even in case of sequential lifecycles, if the project is badly managed and requirements keep changing even when project is in an advanced state, early test implementation can be rendered obsolete.

Therefore, before starting the test implementation process, Test Manager should consider these important points:

- Software development life cycle being used
- Features that need to be tested
- Probability of change in requirement late into project lifecycle
- Possibility of changes in code between two iterations

Test Execution

The term Test Execution tells that the testing for the product or application needs to be executed in order to obtain the expected result.

After the development phase, the testing phase will take place where the various levels of testing techniques will be carried out and the creation and execution of test cases will be taken place.

Test Execution is the process of executing the tests written by the tester to check whether the developed code or functions or modules are providing the expected result as per the client requirement or business requirement.

Importance of Test Execution:

- **The project runs efficiently:** Test execution ensures that the project runs smoothly and efficiently.
- **Application competency:** It also helps to make sure the application's competency in the global market.
- **Requirements are correctly collected:** Test executions make sure that the requirements are collected correctly and incorporated correctly in design and architecture.
- **Application built in accordance with requirements:** It also checks whether the software application is built in accordance with the requirements or not.

Activities for Test Execution

The following are the 5 main activities that should be carried out during the test execution.

1. **Defect Finding and Reporting:** Defect finding is the process of identifying the bugs or errors raised while executing the test cases on the developed code or modules. If any error appears or any of the test cases failed then it will be recorded and the same will be reported to the respective development team.
2. **Defect Mapping:** After the error has been detected and reported to the development team, the development team will work on those errors and fix them as per the requirement. Once the development team has done its job, the tester team will again map the test cases or test scripts to that developed module or code to run the entire tests to ensure the correct output.

3. Re-Testing: Re-Testing is the process of testing the modules or entire product again to ensure the smooth release of the module or product. In some cases, the new module or functionality will be developed after the product release. In this case, all the modules will be re-tested for a smooth release. So that it cannot cause any other defects after the release of the product or application.

4.Regression Testing: Regression Testing is software testing that ensures that the newly made changes to the code or newly developed modules or functions should not affect the normal processing of the application or product.

5.System Integration Testing: System Integration Testing is a type of testing technique that will be used to check the entire component or modules of the system in a single run. It ensures that the whole system will be checked in a single test environment instead of checking each module or function separately.

Test Execution Process

The test Execution technique consists of three different phases. The three main phases of test execution are the creation of test cases, test case execution, and validation of test results.

1. Creation of Test Cases: The first phase is to create suitable test cases for each module or function. Here, the tester with good domain knowledge must be required to create suitable test cases. The creation of test cases should not be delayed else it will cause excess time to release the product. The created test cases should not be repeated again. It should cover all the possible scenarios raised in the application.

2. Test Cases Execution: After test cases have been created, execution of test cases will take place. The Quality Analyst team will either do automated or manual testing depending upon the test case scenario. The selection of testing tools is also important to execute the test cases.

3. Validating Test Results: After executing the test cases, note down the results of each test case in a separate file or report. Check whether the executed test cases achieved the expected result and record the time required to complete each test case i.e., measure the performance of each test case. If any of the test cases is failed or not satisfied the condition then report it to the development team for validating the code.

Ways to Perform Test Execution

1. **Run test cases:** It is a simple and easiest approach to run test cases on the local machine and it can be coupled with other artifacts like test plans, test suites, test environments, etc.
2. **Run test suites:** A test suite is a collection of manual and automated test cases and the test cases can be executed sequentially or in parallel. Sequential execution is useful in cases where the result of the last test case depends on the success of the current test case.
3. **Run test case execution and test suite execution records:** Recording test case execution and test suite execution is a key activity in the test process and helps to reduce errors, making the testing process more efficient.
4. **Generate test results without execution:** Generating test results from non-executed test cases can be helpful in achieving comprehensive test coverage.
5. **Modify execution variables:** Execution variables can be modified in the test scripts for particular test runs.
6. **Run automated and manual tests:** Test execution can be done manually or can be automated.
7. **Schedule test artifacts:** Test artifacts include video, screenshots, data reports, etc. These are very helpful as they document the results of the past test execution and provide information about what needs to be done in future test execution.
8. **Defect tracking:** Without defect tracking test execution is not possible, as during testing one should be able to track the defects and identify what when wrong and where.

Test Execution Priorities are nothing but prioritizing the test cases depending upon several factors.

- **Complexity:** The complexity of the test cases can be determined by including several factors such as boundary values of test cases, features or components of test cases, data entry of test cases, and how much the test cases cover the given business problem.
- **Risk Covered:** How much risk that a certain test case may undergo to achieve the result. Risk in the form of time required to complete the test case process, space complexity whether it is executed in the given memory space, etc.,
- **Platforms Covered:** It simply tells that in which platform or operating system the test cases have been executed i.e., test cases executed in the Windows OS, Mac OS, Mobile OS, etc.,
- **Depth:** It covers how depth the given test cases cover each functionality or module in the application i.e., how much a given test procedure covers all the possible conditions in a single functionality or module.
- **Breadth:** It covers how the breadth of the given test cases covers the entire functionality or modules in the application i.e., how much a given test procedure covers all the possible conditions in the entire functionality or modules in the product or application.

Test Execution States

The tester or the Quality Analyst team reports or notices the result of each test case and records it in their documentation or file. There are various results raised when executing the test cases. They are

- **Pass:** It tells that the test cases executed for the module or function are successful.
- **Fail:** It tells that the test cases executed for the module or function are not successful and resulted in different outputs.
- **Not Run:** It tells that the test cases are yet to be executed.
- **Partially Executed:** It tells that only a certain number of test cases are passed and others aren’t met the given requirement.
- **Inconclusive:** It tells that the test cases are executed but it requires further analysis before the final submission.
- **In Progress:** It tells that the test cases are currently executed.
- **Unexpected Result:** It tells that all the test cases are executed successfully but provide different unexpected results.

Test Execution Report

The Test Execution Report is nothing but a document that contains all the information about the test execution process. The documentation or the report contains various information. They are:

- Who all are going to execute the test cases?
- Who is doing the unit testing, integration testing, system testing, etc.,
- Who is going to write test cases?
- The number of test cases executed successfully.
- The number of test cases failed during the testing.
- The number of test cases executed today.
- The number of test cases yet to be executed.
- What are the automation test tools used for today's test execution?
- What are the modules/functions testing today?
- Recording the issues while executing the test cases.
- What is today's testing plan?
- What is tomorrow's testing plan?

- Recording the pending plans.
- Overall success rate.
- Overall failure rate.
- Test Summary Report Identifier.
- Summary.
- Variances.
- Comprehensive Assessment.
- Summary of Results.
- Evaluation.
- Summary of Activities.
- Approval.

Guidelines for Test Execution

- Write the suitable test cases for each module of the function.
- Assign suitable test cases to respective modules or functions.
- Execute both manual testing as well as automated testing for successful results.
- Choose a suitable automated tool for testing the application.
- Choose the correct test environment setup.
- Note down the execution status of each test case and note down the time taken by the system to complete the test cases.
- Report all the success status and the failure status to the development team
- Track the test status again for the already failed test cases and report it to the team.
- Highly Skilled Testers are required to perform the testing with less or zero failures/defects.
- Continuous testing is required until success test report is achieved.

Reviews, Inspection and Auditing

- https://abodeqa.com/reviewswalkthrough-and-inspection-in-software-testing/

Advantages

- The hybrid method provides features of both Bottom Up and Top Down methods.
- It is most time reducing method.
- It provides complete testing of all modules.

Disadvantages

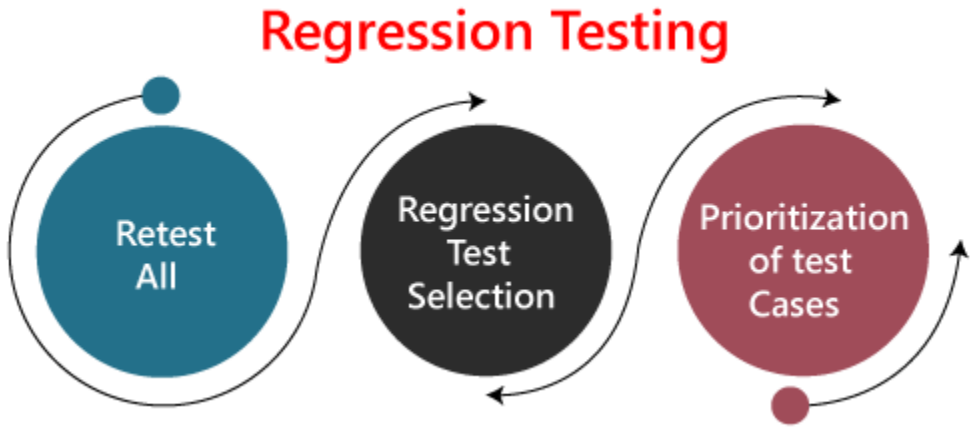
- This method needs a higher level of concentration as the process carried out in both directions simultaneously.
- Complicated method.

Regression Testing

- Regression testing is a black box testing techniques. It is used to authenticate a code change in the software does not impact the existing functionality of the product. Regression testing is making sure that the product works fine with new functionality, bug fixes, or any change in the existing feature.
- Regression testing is a type of software testing. Test cases are re-executed to check the previous functionality of the application is working fine, and the new changes have not produced any bugs.
- Regression testing can be performed on a new build when there is a significant change in the original functionality. It ensures that the code still works even when the changes are occurring. Regression means Re-test those parts of the application, which are unchanged.
- Regression tests are also known as the Verification Method. Test cases are often automated. Test cases are required to execute many times and running the same test case again and again manually, is time-consuming and tedious too.

- When can we perform Regression Testing?
- We do regression testing whenever the production code is modified.
- We can perform regression testing in the following scenario, these are:
- **1. When new functionality added to the application.**
- **Example:**
- A website has a login functionality which allows users to log in only with Email. Now providing a new feature to do login using Facebook.
- **2. When there is a Change Requirement.**
- **Example:**
- Remember password removed from the login page which is applicable previously.
- **3. When the defect fixed**
- **Example:**
- Assume login button is not working in a login page and a tester reports a bug stating that the login button is broken. Once the bug fixed by developers, tester tests it to make sure Login Button is working as per the expected result. Simultaneously, tester tests other functionality which is related to the login button.
-

- **4. When there is a performance issue fix**
- **Example:**
- Loading of a home page takes 5 seconds, reducing the load time to 2 seconds.
- **5. When there is an environment change**
- **Example:**
- When we update the database from MySql to Oracle
- How to perform Regression Testing?
- The need for regression testing comes when software maintenance includes enhancements, error corrections, optimization, and deletion of existing features. These modifications may affect system functionality. Regression Testing becomes necessary in this case.
- Regression testing can be performed using the following techniques:



1. Re-test All:

Re-Test is one of the approaches to do regression testing. In this approach, all the test case suits should be re-executed. Here we can define re-test as when a test fails, and we determine the cause of the failure is a software fault. The fault is reported, we can expect a new version of the software in which defect fixed. In this case, we will need to execute the test again to confirm that the fault fixed. This is known as re-testing. Some will refer to this as confirmation testing.
The re-test is very expensive, as it requires enormous time and resources.

2. Regression test Selection:

•In this technique, a selected test-case suit will execute rather than an entire test-case suit.
•The selected test case suits divided in two cases

- Reusable Test cases.
- Obsolete Test cases.

•Reusable test cases can use in succeeding regression cycle.
•Obsolete test cases can't use in succeeding regression cycle.

- **3. Prioritization of test cases:**
- Prioritize the test case depending on business impact, critical and frequently functionality used. Selection of test cases will reduce the regression test suite.

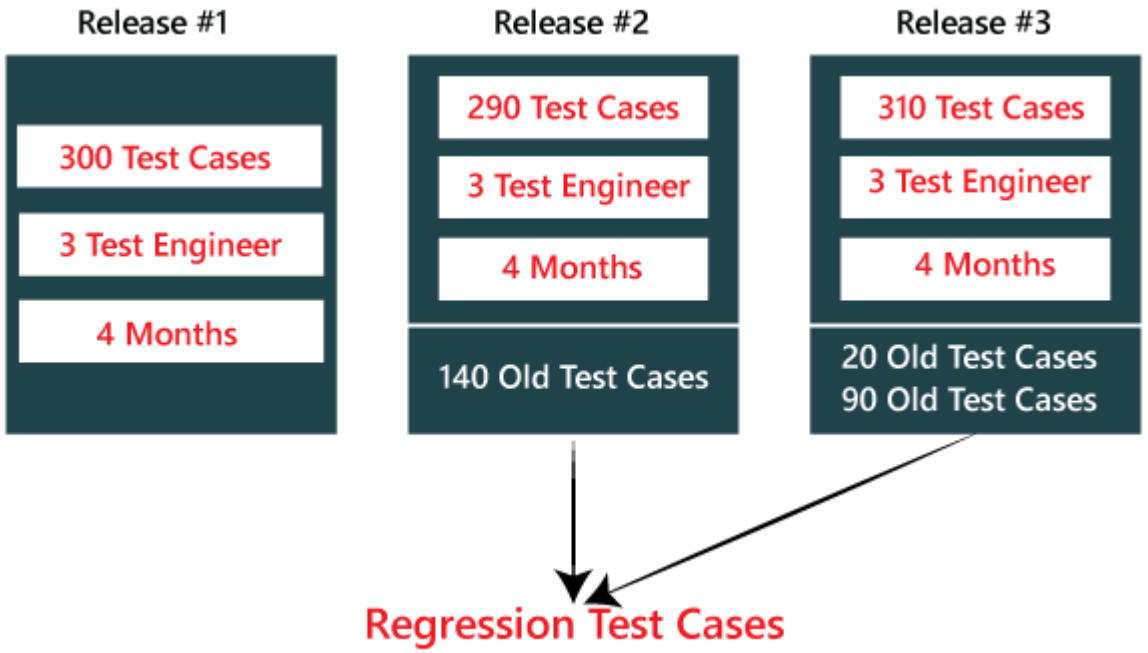
Regression Testing Process

- The regression testing process can be performed across the **builds** and the **releases**.
- Regression testing across the builds
- Whenever the bug fixed, we retest the Bug, and if there is any dependent module, we go for a Regression Testing.

- Regression Testing Across the Release
- The regression testing process starts whenever there is a new Release for same project because the new feature may affect the old elements in the previous releases.
- To understand the regression testing process, we will follow the below steps:
- **Step1**
- There is no regression testing in **Release#1** because there is no modification happen in the Release#1 as the release is new itself.
- **Step2**
- The concept of Regression testing starts from **Release#2** when the customer gives some **new requirements**.
- **Step3**
- After getting the new requirements (modifying features) first, they (the developers and test engineers) will understand the needs before going to the **impact analysis**.
- **Step4**
- After understanding the new requirements, we will perform one round of **impact analysis** to avoid the major risk, but here the question arises who will do the Impact analysis?
- **Step5**
- The impact analysis is done by the **customer** based on their **business knowledge**, the **developer** based on their **coding knowledge**, and most importantly, it is done by the **test engineer** because they have the **product knowledge**.

- **Step6**
- Once we are done with the **impact area**, then the developer will prepare the **impact area (document)**, and the **customer** will also prepare the **impact area document** so that we can achieve the **maximum coverage of impact analysis**.
- **Step7**
- After completing the impact analysis, the developer, the customer, and the test engineer will send the **Reports#** of the impact area documents to the **Test Lead**. And in the meantime, the test engineer and the developer are busy working on the new test case.
- **Step8**
- Once the Test lead gets the Reports#, he/she will **consolidate** the reports and stored in the **test case requirement repository** for the release#1.

- **Step9**
- After that, the Test Lead will take the help of RTM and pick the necessary **regression test case** from the **test case repository**, and those files will be placed in the **Regression Test Suite**.
- **Note:**
- The test lead will store the regression test case in the regression test suite for no further confusion.
- **Regression test suite**: Here, we will save all the impact area test documents.
- **Regression Test Cases**: These are the test cases of the old releases text document which need to be re-executed as we can see in the below image:

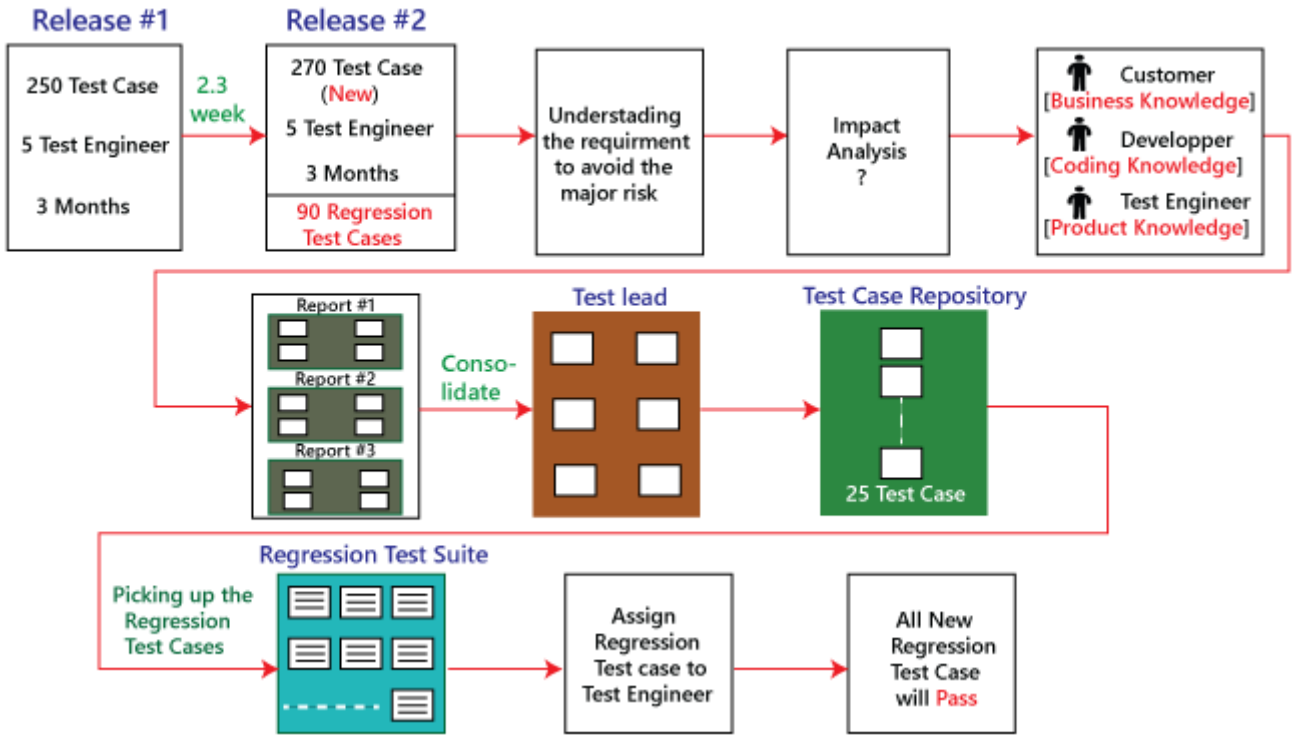


Step10

After that, when the test engineer has done working on the new test cases, the test lead will **assign the regression test case** to the test engineer.

Step11

When all the regression test cases and the new features are **stable and pass**, then check the **impact area using the test case** until it is durable for old features plus the new features, and then it will be handed over to the customer.

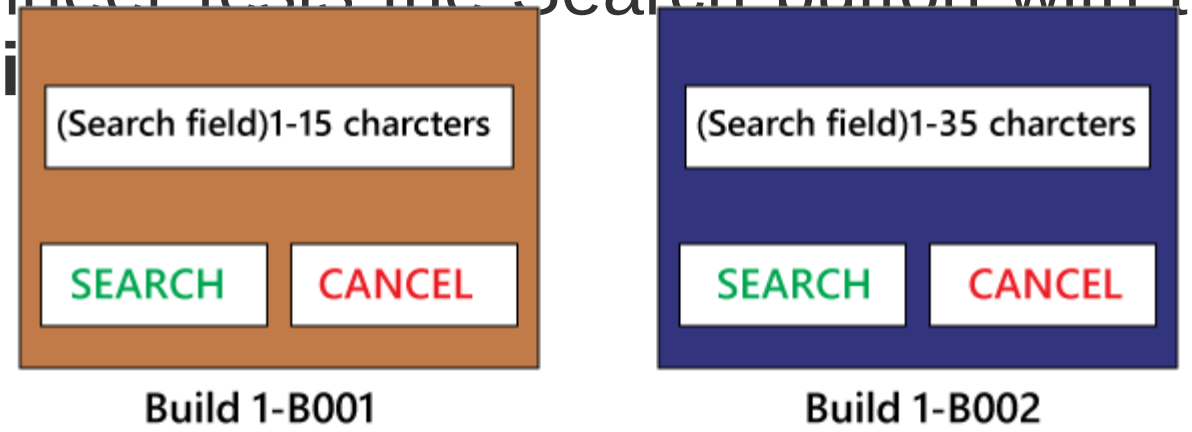


Types of Regression Testing

The different types of Regression Testing are as follows:

1.Unit Regression Testing [URT]
2.Regional Regression Testing[RRT]
3.Full or Complete Regression Testing [FRT]

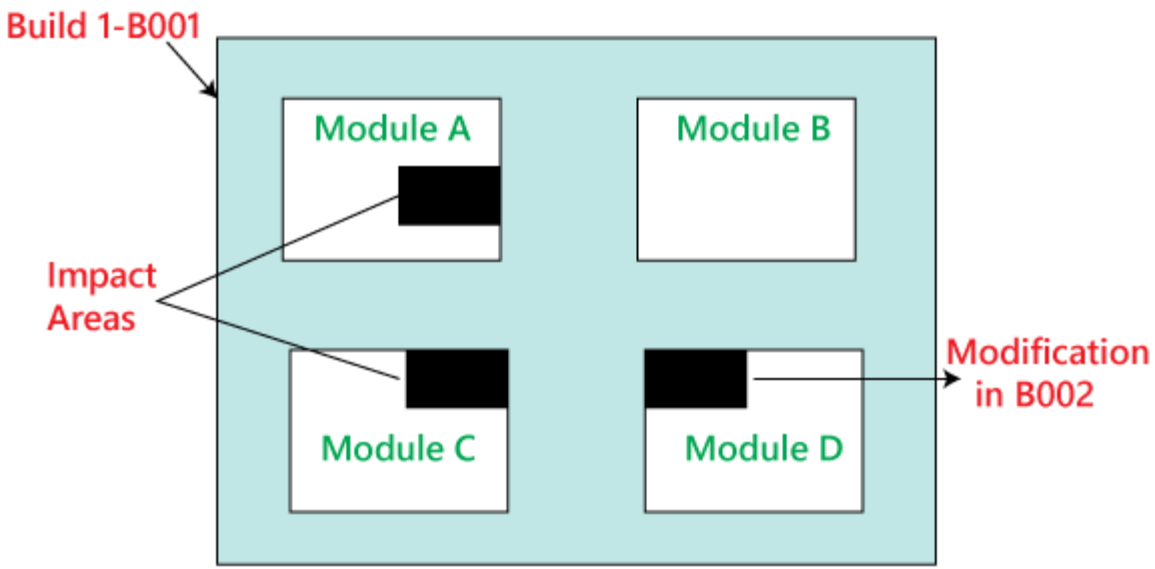
- 1) Unit Regression Testing [URT]
- In this, we are going to test only the changed unit, not the impact area, because it may affect the components of the same module.
- **Example1**
- In the below application, and in the first build, the developer develops the **Search** button that accepts **1-15 characters**. Then the test engineer tests the Search button with the help of the **test case desi**
-



- Now, the client does some modification in the requirement and also requests that the **Search button** can accept the **1-35 characters**. The test engineer will test only the Search button to verify that it takes 1-35 characters and does not check any further feature of the first build.
- **Example2**
- Here, we have **Build B001**, and a defect is identified, and the report is delivered to the developer. The developer will fix the bug and sends along with some new features which are developed in the second **Build B002**. After that, the test engineer will test only after the defect is fixed.
- The test engineer will identify that clicking on the **Submit** button goes to the blank page.
- And it is a defect, and it is sent to the developer for fixing it.
- When the new build comes along with the bug fixes, the test engineer will test only the Submit button.
- And here, we are not going to check other features of the first build and move to test the new features and sent in the second build.
- We are sure that fixing the **Submit**button is not going to affect the other features, so we test only the fixed bug.

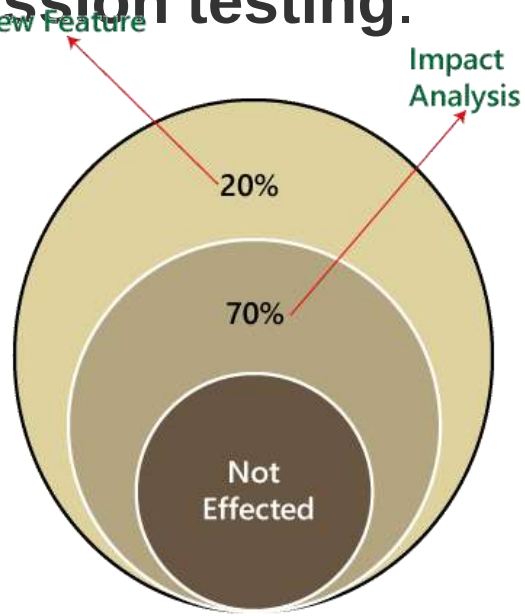
Therefore, we can say that by testing only the changed feature is called the **Unit Regression Testing**.

- 2) Regional Regression testing [RRT]
- In this, we are going to test the modification along with the impact area or regions, are called the **Regional Regression testing**. Here, we are testing the impact area because if there are dependable modules, it will affect the other modules also.
- **For example:**
- In the below image as we can see that we have four different modules, such as **Module A, Module B, Module C, and Module D**, which are provided by the developers for the testing during the first build. Now, the test engineer will identify the bugs in **Module D**. The bug report is sent to the developers, and the development team fixes those defects and sends the second build.



In the second build, the previous defects are fixed. Now the test engineer understands that the bug fixing in Module D has impacted some features in **Module A and Module C**. Hence, the test engineer first tests the Module D where the bug has been fixed and then checks the impact areas in **Module A and Module C**. Therefore, this testing is known as **Regional regression testing.**

- 3) Full Regression testing [FRT]
- During the second and the third release of the product, the client asks for adding 3-4 new features, and also some defects need to be fixed from the previous release. Then the testing team will do the Impact Analysis and identify that the above modification will lead us to test the entire product.
- Therefore, we can say that testing the **modified features** and **all the remaining (old) features** is called the **Full Regression testing**.



- When we perform Full Regression testing?
- We will perform the FRT when we have the following conditions:
- When the modification is happening in the source file of the product. **For example**, JVM is the root file of the JAVA application, and if any change is going to happen in JVM, then the entire JAVA program will be tested.
- When we have to perform n-number of changes.

`

- Mutation Testing
- Mutation testing is a white box method in software testing where we insert errors purposely into a program (under test) to verify whether the existing test case can detect the error or not. In this testing, the mutant of the program is created by making some modifications to the original program.
- The primary objective of mutation testing is to check whether each mutant created an output, which means that it is different from the output of the original program. We will make slight modifications in the mutant program because if we change it on a massive scale than it will affect the overall plan.
- When we detected the number of errors, it implies that either the program is correct or the test case is inefficient to identify the fault.
- Mutation testing purposes is to evaluate the quality of the case that should be able to fail the mutant code hence this method is also known as Fault-based testing as it used to produce an error in the program and that why we can say that the mutation testing is performed to check the efficiency of the test cases.

Types of mutation testing

- Mutation testing can be classified into three parts, which are as follows:
- Decision mutations
- value mutations
- Statement mutations

Decision mutations

- In this type of mutation testing, we will check the design errors. And here, we will do the modification in arithmetic and logical operator to detect the errors in the program.
- Like if we do the following changes in arithmetic operators:
- plus(+)→ minus(-)
- asterisk(*)→ double asterisk(**)
- plus(+)→incremental operator(i++)
- Like if we do the following changes in logical operators
- Exchange P **>** → P**<**, OR P**>=**
- Now, let see one example for our better understanding:

Original Code	Modified Code

if(p >q)
r= 5;
else
r= 15;
	if(p < q)
	r = 5;
	else
	r = 15;

Value mutations

In this, the values will modify to identify the errors in the program, and generally, we will change the following:

- Small value à higher value
- Higher value àSmall value.

For Example:

Original Code	Modified Code

int add =9000008;
int p = 65432;
int q =12345;
int r = (p+ q);
	int mod = 9008;
	int p = 65432;
	int q =12345;
	int r= (p + q);

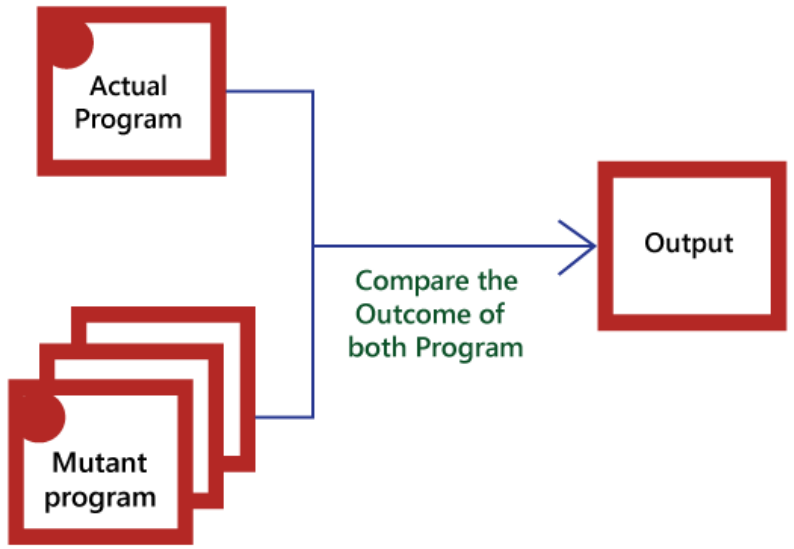
- Statement Mutations
- Statement mutations means that we can do the modifications into the statements by removing or replacing the line as we see in the below example:

Original Code	Modified Code

if(p * q)
r = 15;
else
r = 25;
	if(p* q)
	s = 15;
	else
	s = 25;

How to perform mutation testing

- To perform mutation testing, we will follow the below process:



- In this, firstly, we will add the errors into the source code of the program by producing various versions, which are known mutants. Here every mutant having the one error, which leads the mutant kinds unsuccessful and also validates the efficiency of the test cases.
- After that, we will take the help of the test cases in the mutant program and the actual application will find the errors in the code.
- Once we identify the faults, we will match the output of the actual code and mutant code.
- After comparing the output of both actual and mutant programs, if the results are not matched, then the mutant is executed by the test cases. Therefore the test case has to be sufficient for identifying the modification between the actual program and the mutant program.
- And if the actual program and the mutant program produced the exact result, then the mutant is saved. And those cases are more active test cases because it helps us to execute all the mutants.

- **Web Based Testing**
- Web testing is a software testing technique to test web applications or websites for finding errors and bugs. A web application must be tested properly before it goes to the end-users. Also, testing a web application does not only means finding common bugs or errors but also testing the quality-related risks associated with the application. Software Testing should be done with proper tools and resources and should be done effectively. We should know the architecture and key areas of a web application to effectively plan and execute the testing.
- Testing a web application is very common while testing any other application like testing functionality, configuration, or compatibility, etc. Testing a web application includes the analysis of the web fault compared to the general software faults. Web applications are required to be tested on different browsers and platforms so that we can identify the areas that need special focus while testing a web application.

- **Types of Web Testing:**
- Basically, there are 4 types of web-based testing that are available and all four of them are discussed below:
- **Static Website Testing:** A static website is a type of website in which the content shown or displayed is exactly the same as it is stored in the server. This type of website has great UI but does not have any dynamic feature that a user or visitor can use. In static testing, we generally focus on testing things like UI as it is the most important part of a static website. We check things font size, color, spacing, etc. testing also includes checking the contact us form, verifying URLs or links that are used in the website, etc.
- **Dynamic Website Testing:** A dynamic website is a type of website that consists of both frontend i.e, UI, and the backend of the website like a database, etc. This type of website gets updated or change regularly as per the user's requirements. In this website, there are a lot of functionalities involved like what a button will do if it is pressed, are error messages are shown properly at their defined time, etc. We check if the backend is working properly or not, like does entering the data or information in the GUI or frontend gets updated in the databases or not.
- **E-Commerce Website Testing:** An e-commerce website is very difficult in maintaining as it consists of different pages and functionalities, etc. In this testing, the tester or developer has to check various things like checking if the shopping cart is working as per the requirements or not, are user registration or login functionality is also working properly or not, etc. The most important thing in this testing is that does a user can successfully do payment or not and if the website is secured. And there are a lot of things that a tester needs to test apart from the given things.
- **Mobile-Based Web Testing:** In this testing, the developer or tester basically checks the website compatibility on different devices and generally on mobile devices because many of the users open the website on their mobile devices. So, keeping that thing in mind, we must check that the site is responsive on all devices or platforms.

Points to be Considered While Testing a Website:

- Do all pages are having valid internal and external links or URLs?
- Whether the website is working as per the system compatibility?
- As per the user interface- Does the size of displays are the optimal and the best fit for the website?
- What type of security does the website need (if unsecured)?
- What are the requirements for getting the website analytics, and also controlling graphics, URLs, etc?
- The contact us or customer assistance feature should be added or not on the page, and etc?

Mobile App Testing Tools

- **Best Mobile App Testing Tools for Automation Testing**
- **1) Kobiton**
- Kobiton allows testers an easy-to-use platform to access real devices for manual and automated testing. Kobiton supports complex gestures, ADB shell commands, geo-location, and device connection management. It also offers real-time insight into logs users can explore and download so issues can be identified and resolved.

- **Features:**
- Latest real, cloud-based devices and configurations
- Centralized testing history and data logs for increased collaboration
- Internal Device Lab Management to more effectively utilizes internal devices
- Simplified user experience to streamline test sessions
- On-premises deployment option for extra security
- Support programming language like C#, Java, Ruby, NodeJS, PHP, and Python
- Offers different framework like React Native, Ionic, Electron NativeScript, Xamarin, and Flutter
- Supports Performance Testing, Automation testing, Manual Testing, Functional Testing, and more
- Seamlessly integrates with Travis CI, TeamCity, Jenkins, Azure DevOps, XebiaLabs, Circleci, Jira, and more
- Provides Record-and-replay, Cross-browsing functionality, No-code automation, and real devices testing
- Offers Capabilities, Performance, Customizable Test Cloud, Rich Test Logs, Agile Test Enabler, and Optimized Efficiency
- It provides customer support via Chat, Contact form, and Email
- **Supported Platforms:** iOS, and Android
- **Price:** Plans start at $75 a month.
- **Free Trial:** 14 Days Free Trial (No Credit Card Required)

- 2) testRigor
- testRigor helps you to directly express tests as executable specifications in plain English. Users of all technical abilities are able to build end-to-end tests of any complexity covering mobile, web and API steps in one test. Test steps are expressed on the end-user level instead of relying on details of implementation like XPaths or CSS Selectors.

- **Features:**
- Test cases are in English
- Unlimited users & Unlimited tests
- The easiest way to learn automation
- Recorder for web steps
- Email & SMS testing
- Web + Mobile + API steps in one test
- Support programming language like Python, Java, Ruby, JavaScript, PHP, and C#
- Offers different framework like Android, iOS, Angular, React, React Native, and Flutter
- Supports API Testing, Audio Testing, Functional Testing, Security Testing, and more
- Seamlessly integrates with TestRail, Zephyr, XRay, Jira, Azure DevOps, Jenkins, CircleCI, Azure DevOps, and more
- Provides Record-and-replay, Cross-browsing functionality, and No-code automation
- Offers Mocking of API calls, Accessing DBs, 2FA login support, Validation of downloaded files, Generate unique test data, multiple browsers and devices and Reusable Rules
- It provides customer support via Contact Form
- **Supported Platforms:** iOS, and Android
- **Price:** Plans start at $900 a month.
- **Free Trial:** Lifetime Free Public Open Source Plan

- 3) ACCELQ
- ACCELQ offers AI-powered codeless test automation and management built on a cloud-native platform. ACCELQ provides a unified platform for mobile, web, API, database, and packaged apps. Automation-first, codeless capabilities make it easy to use for testing teams without deep programming expertise. ACCELQ allows businesses to achieve 3x productivity and over 70% savings with its industry-first autonomics-based automation platform.

- **Features:**
- Design, develop and execute mobile test automation with zero setup and no coding.
- Integrated Mobile Cloud Device execution farm with public and private device options for cross-device testing in Plug & Play model.
- Mobile, Web, API, backend and full stack automation in the same unified flow.
- Automation flow recorder, coupled with powerful Natural Language no-code editor.
- Automation that executes across Mobile OS and agnostic of Development frameworks.
- Design-first approach with inbuilt modularity; no need for custom frameworks.
- Robust and sustainable Automation that's significantly low on maintenance.
- Enable Manual testers to Automate without need for programming skills.
- AI-powered Mobile Object handling and self-healing capabilities eliminates Test Flakiness.
- In-sprint automation to align with DevOps and Agile.
- Visual application model for business process validation.
- AI based Automated test case generation and data planning.
- Built-in test management , Version control and governance capabilities.
- Seamless CI/CD integration and natural traceability.
- **Supported Platforms:** Web on Mobile, Native iOS, Native Android , Hybrid Apps.
- **Price:** Plans start at $150 a month.
- **Free Trial:** 14 Days Free Trial (No Credit Card Required)

- **4) Katalon Platform**
- Built on top of Appium and Selenium, Katalon Platform removes the tools' existing steep learning curve and in turn brings a codeless testing experience to users at all scales and expertise. In addition to supporting Android & IOS platforms, testing across OS (Windows, macOS, and Linux) is also available.

- **Features:**
- Simple setup and effortless test creation using record & playback, keywords, images.
- Execute tests locally and remotely on real devices, simulators, or custom cloud-based devices (Sauces Lab, Kobiton, Perfecto, Lambda Test, etc.)
- Flexible test reusability across mobile platforms, API, and Web.
- Reduce maintenance efforts via built-in integration with commonly used project management tools (Jira, Git, Jenkins, etc.)
- Provide insightful test reports of all testing stages for better monitor and collaboration across teams.
- Support programming language like JavaScript, Python, VBScript, JScript, Delphi, C++, and C#
- Offers different framework like Flutter, Xamarin, and React Native
- Supports Visual Testing, Web Testing, API Testing, Mobile Testing, Desktop Testing, and more
- Seamlessly integrates with Azure DevOps, BrowserStack, Circle CI, CodeMagic, Curiosity, GitHub, Google Cloud Build, and more
- Provides Record-and-replay, Cross-browsing functionality, No-code automation, and real devices testing
- Offers Self-Healing, Autonomous Script Generation, Speed, Flexibility & Scalability, Visibility, Innovation & AI and Affordability & ROI
  It provides customer support via Chat, Contact form, and Email
- **Supported Platforms:** iOS, and Android
- **Price:** Plans start at $29 a month. 16% Discount on Yearly Payment.
- **Free Trial:** Lifetime Free Public Open Source Plan

- **5) Perfecto**
- Perfecto is the industry-leading testing cloud for mobile app testing. Prepare your apps for a mobile-first world. Deliver exceptional digital experiences faster and with confidence with Perfecto.

- **Features:**
- Unmatched coverage across platforms and testing scenarios.
- Smart analytics for faster feedback and fixes.
- Unified cloud platform for web and mobile app testing.
- Same-day access to new devices, OSes, and more.
- Enterprise-grade security and scalability.
- Deep technical expertise and support to help you succeed.
- Support programming language like Java, JavaScript, C#, Python, and PHP
- Offers different framework like Flutter, Xamarin, and React Native
- Supports Continuous Testing, Automated Testing, Interactive Testing, Mobile Application Testing, Web Testing, and more
- Seamlessly integrates with Eclipse, IntelliJ, Appium, Espresso, Quantum, TestNG, JUnit, Jira, slack, Perfecto, and more
- Provides Record-and-replay, Cross-browsing functionality, No-code automation, and real devices testing
- Offers Visual Creation, Intelligent Test Creation & Maintenance, Cloud-Based Collaboration, Automatic Device Cleanup, Intelligent Reporting & Debugging, Live Session Sharing, and Secure Local Tunneling
- It provides customer support via Chat, and Contact Form
- **Supported Platforms:** iOS, and Android
- **Price:** Plans start at $99 a month. 16% Discount on Yearly Payment.
- **Free Trial:** 14 Days Free Trial

- 6) TestGrid
- TestGrid allows users to perform both manual and automated testing of their mobile applications on real devices hosted on-cloud or on your premise in the easiest way.
- You can engage your testing and business teams to build and execute test cases without any pre-requisites of programming knowledge. With TestOS users don't have to worry about rewriting different test cases but reuse almost all the tests on different versions of the app and on other apps as well. Start with a free plan and upgrade at as low as $29/mo.

- **7) Appium (iOS/Android Testing Tool)**
- Appium is an open source, and a cross platform Mobile Testing Tool for the hybrid and native iOS, it supports Android versions from 2.3 onwards. Appium works like a server running in the background like selenium server. This mobile automation testing tool supports many programming languages, such as Java, Ruby, C# and other which are in the WebDriver library. Appium utilizes WebDriver interface for tests running
- Appium automates Android using the UIAutomator library, which is given by Google as part of the Android SDK. On mobile devices, it can control Safari and Chrome. It can be synchronized with testing framework TestNG. In this case, UI Automator can produce informative and detailed reports, similar to reports generated by Ranorex

- **8) Selendroid**
- Selendroid is a test automation framework that drives off the UI of Android native and hybrid applications (apps) and the mobile web. Using the Selenium 2 client API tests are written.
- **9) Calabash**
- Calabash consists of libraries that allow test-code to programmatically interact with native and hybrid apps.

10) KIF

- KIF mobile app testing tool is objective C based framework and is purely for iOS automated testing. Kif is an mobile automation framework which integrates directly with XCTests. It can be used when business folk are not involved in writing or reading test specs.

- **What Is DevOps?**
- The goal of DevOps is to build better, faster and more responsive software by bringing Development and Operations teams together. DevOps is not a methodology or a suite of tools but a cultural shift to remove the barriers between Dev and Ops in order to meet the need for shorter and more frequent software deliveries. This will allow your organization to respond in a more agile manner to changing business requirements. The DevOps cultural shift depends on continuously optimizing workflow, architecture, and infrastructure in order to deliver high-quality applications.

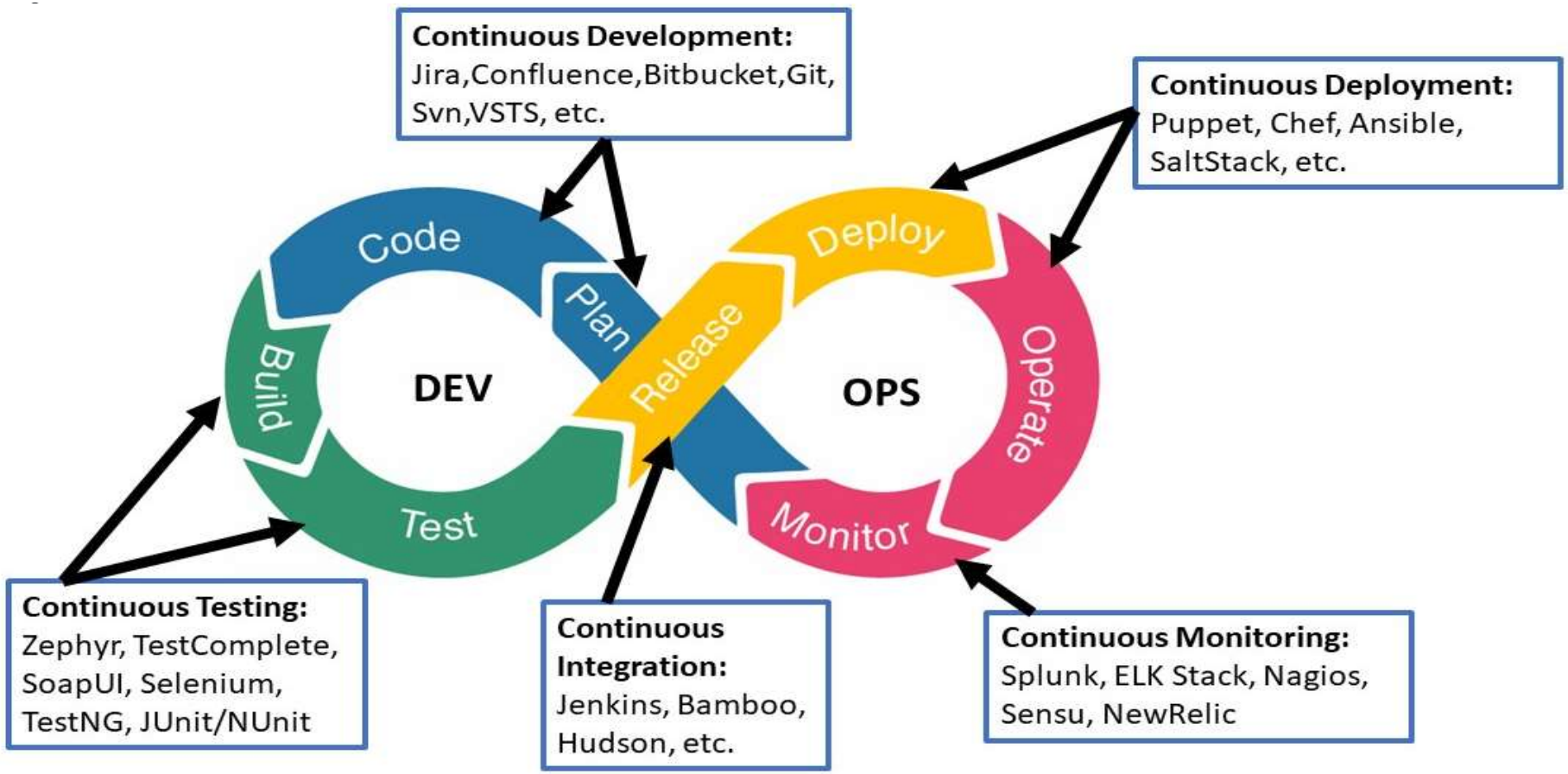
- **From the Agile Model to the DevOps Model**
- Software developed with the Agile Model adheres to the following four basic priorities stated in The Agile Manifesto:

1. Individuals and interactions over processes and tools;
2. Working software over comprehensive documentation;
3. Customer collaboration over contract negotiation; and
4. Responding to change over following a plan.

- Agile development takes a test-first approach, rather than the test-at-the-end approach of traditional development. In agile testing, code is developed and tested in small increments of functionality. Agile project management promotes frequent interaction between an IT department and business users and tries to regularly build and deliver software that meets business users' changing requirements. The flexible approach of the Agile Model helps to ensure a streamlined software development process that responds quickly to customer demand in a wide variety of use cases.

- Agile software development can be implemented with a number of methodologies, including Scrum, Kanban, Scrumban (a mix of Scrum and Kanban), extreme programming, and many more. In the Scrum methodology, the development effort is broken down into a series of short 'sprints,' or iterations, typically lasting 2 to 4 weeks. The goal of each sprint is to create potentially shippable software by adding new or enhancing existing application features.
- The DevOps Model is the 'Agile Model on steroids,' which extends the cross-functional Agile team made up of software designers, testers and developers to include support people from Operations. The focus of this expanded agile team is the overall service or software your company delivers to its customers, instead of just "working software" (the 2nd of the 4 Agile principles listed above).
- Building a successful continuous, two-way DevOps software pipeline between your company and your customers means creating a DevOps culture of collaboration among the various teams involved in software delivery (developers, operations, quality assurance, business analysts, management, and so forth), as well as reducing the cost, time, and risk of delivering software changes by allowing for more incremental updates to applications in production. In practice, this means teams produce software in short cycles, ensuring that the software can be reliably released at any time. Where regular agile teams strive to deliver shippable software in 2 to 4 weeks, the ideal DevOps goal is to deliver code to production daily or every few hours.

- **DevOps Tools**



- software applications pass through five different stages when developed in a DevOps pipeline:

1. Continuous Development
2. Continuous Testing
3. Continuous Integration
4. Continuous Delivery
5. Continuous Monitoring

- Because DevOps is a cultural shift that promotes collaboration between development, QA and operations, there is no one single product that can be considered the definitive DevOps tool. Often a collection of tools, from a variety of vendors, are used in one or more stages of the DevOps toolchain. The DevOps tools used in each stage will vary from organization to organization, based on tools already in place, the skill level of your developers and testers, and the types of applications you're deploying.
- Here is a short list of some the most common DevOps tools and an explanation of how they're used in each stage:

- **Continuous Development with Jira**
- The de facto standard for the agile DevOps software development and testing, Jira Software has powerful collaborative functionality that visually highlights issues as they move through the project workflow, which makes the Jira platform easy to use for planning development, tracking daily work and reporting on project progress.
- **Continuous Testing with Zephyr for Jira**
- Although Jira Software is designed for issue, project and workflow tracking on IT projects, many DevOps teams are also using it for test case management so that development and testing teams can work together in one system. Test issues can be created, executed, tracked and reported on just like any other Jira issue in Zephyr for Jira, which has the same look n' feel of Jira.

- **Continuous Integration with Jenkins**
- Jenkins is a CI/CD server that runs tests automatically every time a developer pushes new code into the source repository. Because CI detects bugs early on in development, bugs are typically smaller, less complex and easier to resolve. Originally created to be a build automation tool for Java applications, Jenkins has since evolved into a multi-faceted platform with over a thousand plug-ins for other software tools. Because of the rich ecosystem of plug-ins, Jenkins can be used to build, deploy and automate almost any software project--regardless of the computer

- **Continuous Delivery/Deployment with Puppet or Chef**
- Automated configuration tools such as Puppet or Chef allow you to avoid the problem of “snowflake servers" in your delivery/deployment environment. “Snowflake servers" are long-running production servers that have been repeatedly reconfigured and modified over time with operating system upgrades, kernel patches, and/or system updates to fix security vulnerabilities. This leads to a server that is unique and distinct, like a snowflake, which requires manual configuration above and beyond that covered by automated deployment scripts.
- Using definition files (called manifests in Puppet, or recipes in Chef) to describe the configuration of the elements of a server is a key best practice in Infrastructure as Code, which allows environments (machines, network devices, operating systems, middleware, etc.) to be configured and specified in a fully automatable format rather than by physical hardware configuration or the use of interactive configuration tools.

- **Continuous Monitoring with Splunk or Elk**
- Both Spunk and Elk are log monitoring tools that allow you to do data analysis on transactions that take place in your IT applications after they've been deployed to ensure uniform performance, availability, security, and user experience. Both Splunk and ELK supply a scalable way to collect and index log files and provide a search interface for users to interact with the data in order to create visualizations such as reports, dashboards or alerts.
- Splunk is a paid service that bills according to the volume of your transactions. The ELK Stack is a set of three open-source products—Elasticsearch, Logstash, and Kibana—all developed and maintained by Elastic.

- **DevOps Test Automation**
- Choosing the right DevOps test automation tool is vitally important for your organization since the right software testing tool in the right hands can foster team collaboration, drive down costs, shorten release cycles, and provide real-time visibility into the status and quality of applications in your DevOps pipeline.
- Test automation works by running a large number of tests repeatedly to make sure an application doesn't break whenever new changes are introduced—at the different Unit-, API- and GUI-levels of the Test Automation Pyramid. In addition to Continuous Integration tools such as Jenkins described above, here are some other useful tools for building, testing and deploying applications automatically when requirements change in order to speed up the release velocity of your pipeline apps.

- **Bamboo**
- Bamboo is a CI/CD server from Atlassian. Like Jenkins and other CI/CD servers, Bamboo allows developers to automatically build, integrate, test and deploy source code. Bamboo is closely connected with other Atlassian tools such as Jira for project management and HipChat for team communication. Unlike Jenkins, which is a free and open source agile automation tool, Bamboo is commercial software that is integrated (and supported) out of the box with other Atlassian products such as Bitbucket, Jira, and Confluence.
- **Selenium**
- Selenium is a suite of different open-source software tools used for automated software testing of web applications across various browsers/platforms. Most often used to create robust, browser-based regression automation suites and tests, Selenium--like Jenkins--has a rich repository of open source tools that are useful for different kinds of automation problems. With support for programming languages like C#, Java, JavaScript, Python, Ruby, .Net, Perl, PHP, etc., Selenium can be used to write automation scripts that run against most modern web browsers.

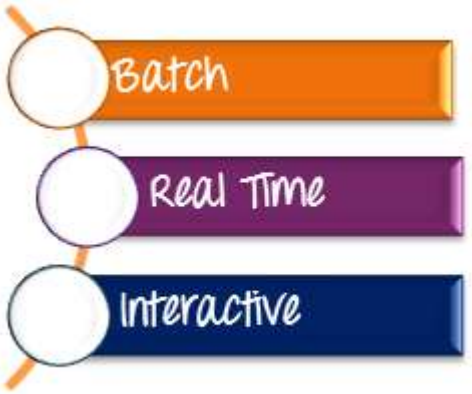
- **TestComplete**
- TestComplete has a powerful and comprehensive set of features for web, mobile, and desktop application testing. In addition to having an easy-to-use record and playback feature, TestComplete allows testers to use JavaScript, VBScript, Python, or C++Script to write test scripts. The tool also an object recognition engine that can accurately detect dynamic user interface elements, which makes it especially useful in applications that have dynamic and frequently changing user interfaces. Since it's a SmartBear product, TestComplete can be integrated easily with other products offered by SmartBear.
- **SoapUI**
- SoapUI is a test automation tool for functional testing, web services testing, security testing, and load testing. Specifically designed for API testing, SoapUI supports both REST and SOAP services. SoapUI provides drag and drops options for creating test suites, test steps and test requests to build complex test scenarios without having to write scripts.

Big Data Testing

- **Big Data Testing** is a testing process of a big data application in order to ensure that all the functionalities of a big data application works as expected. The goal of big data testing is to make sure that the big data system runs smoothly and error-free while maintaining the performance and security.
- Big data is a collection of large datasets that cannot be processed using traditional computing techniques. Testing of these datasets involves various tools, techniques, and frameworks to process. Big data relates to data creation, storage, retrieval and analysis that is remarkable in terms of volume, variety, and velocity. You can learn more about Big Data, Hadoop and MapReduce

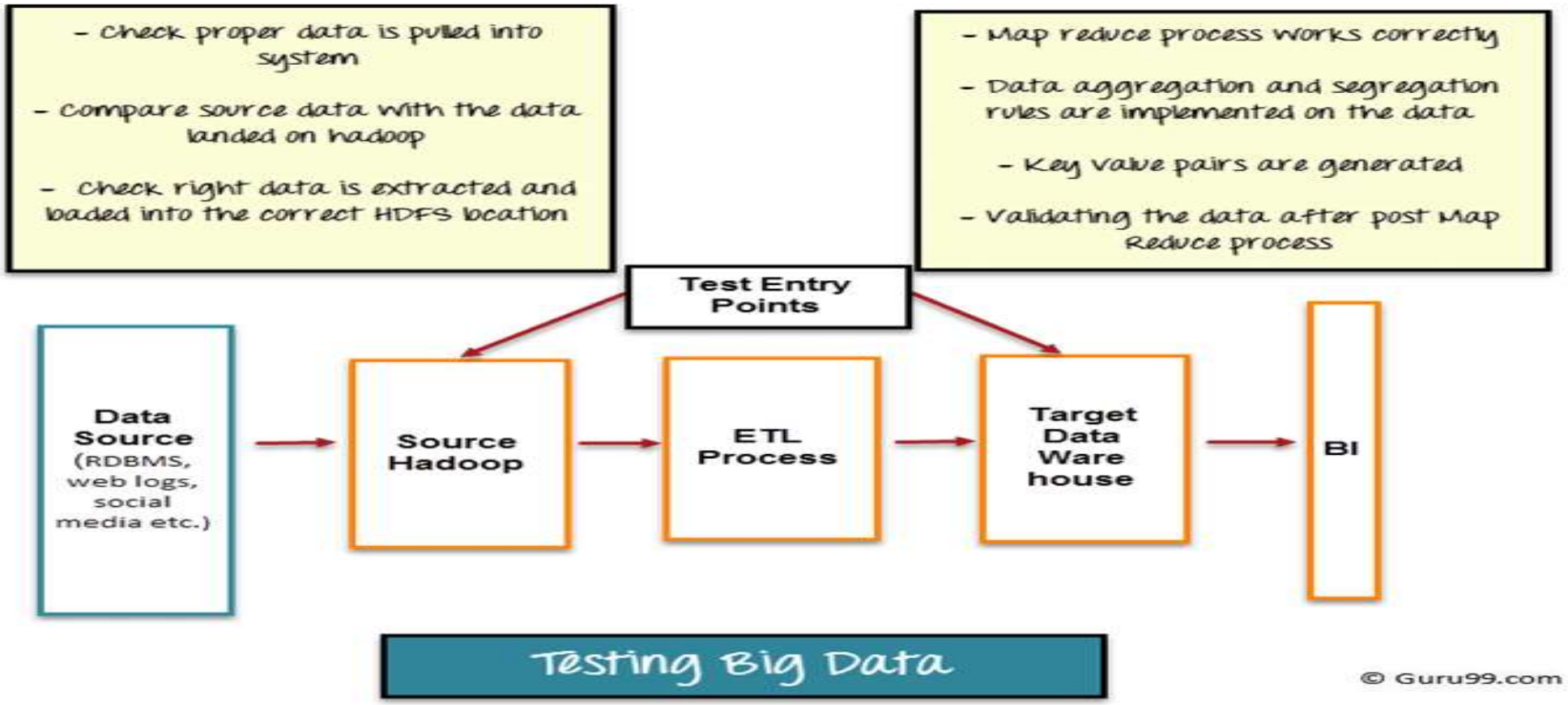
What is Big Data Testing Strategy?

- Testing Big Data application is more verification of its data processing rather than testing the individual features of the software product. When it comes to Big data testing, **performance and functional testing** are the keys.
- In Big Data testing strategy, QA engineers verify the successful processing of terabytes of data using commodity cluster and other supportive components. It demands a high level of testing skills as the processing is very fast. Processing may be of three types



- Along with this, data quality is also an important factor in Hadoop testing. Before testing the application, it is necessary to check the quality of data and should be considered as a part of database testing. It involves checking various characteristics like conformity, accuracy, duplication, consistency, validity, data completeness, etc. Next in this Hadoop Testing tutorial, we will learn how to test Hadoop applications.

- **How to test Hadoop Applications**
- The following figure gives a high-level overview of phases in Testing Big Data Applications
-



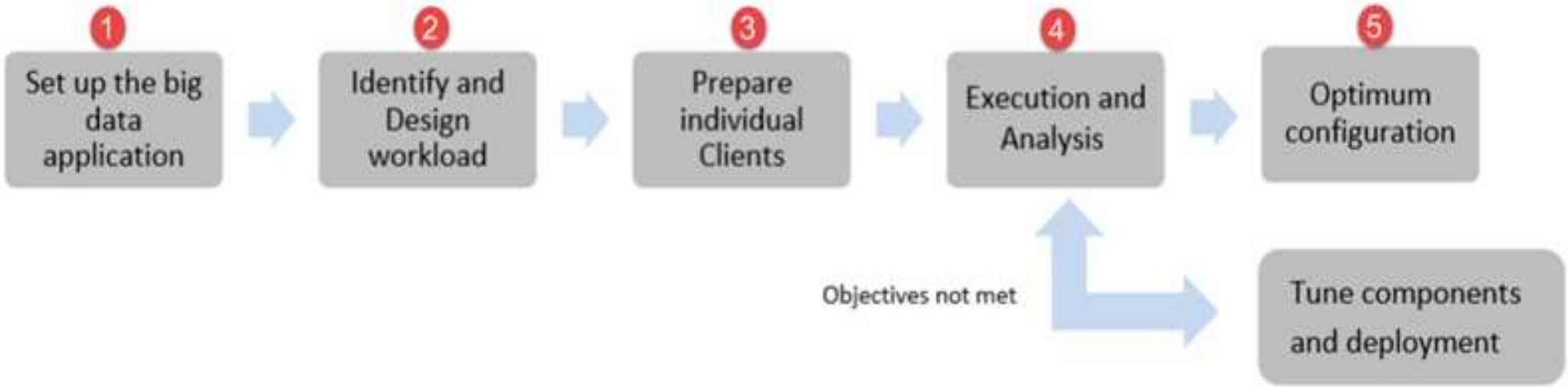
- Big Data Testing or Hadoop Testing can be broadly divided into three steps
- **Step 1: Data Staging Validation**
- The first step in this big data testing tutorial is referred as pre-Hadoop stage involves process validation.
- [illegible], weblogs, social media, etc. should be validated to make sure that correct data is pulled into the system
- Comparing source data with the data pushed into the Hadoop system to make sure they match
- Verify the right data is extracted and loaded into the correct HDFS location
- Tools like Talend, Datameer, can be used for data staging validation
- **Step 2: "MapReduce" Validation**
- [illegible]ness logic validation on every node and then validating them after running against multiple nodes, ensuring that the
- Map Reduce process works correctly
- Data aggregation or segregation rules are implemented on the data
- Key value pairs are generated
- Validating the data after the Map-Reduce process

- **Step 3: Output Validation Phase**
- The final or third stage of Hadoop testing is the output validation process. The output data files are generated and ready to be moved to an EDW (Enterprise Data Warehouse) or any other system based on the requirement.
- Activities in the third stage include
- To check the transformation rules are correctly applied
- To check the data integrity and successful data load into the target system
- To check that there is no data corruption by comparing the target data with the HDFS file system data

- **Architecture Testing**
- Hadoop processes very large volumes of data and is highly resource intensive. Hence, architectural testing is crucial to ensure the success of your Big Data project. A poorly or improper designed system may lead to performance degradation, and the system could fail to meet the requirement. At least, **Performance and Failover test** services should be done in a Hadoop environment.
- Performance testing includes testing of job completion time, memory utilization, data throughput, and similar system metrics. While the motive of Failover test service is to verify that data processing occurs seamlessly in case of failure of data nodes

- **Performance Testing**
- Performance Testing for Big Data includes two main action
- **Data ingestion and Throughout:** In this stage, the Big Data tester verifies how the fast system can consume data from various data source. Testing involves identifying a different message that the queue can process in a given time frame. It also includes how quickly data can be inserted into the underlying data store for example insertion rate into a Mongo and Cassandra database.
- **Data Processing:** It involves verifying the speed with which the queries or map reduce jobs are executed. It also includes testing the data processing in isolation when the underlying data store is populated within the data sets. For example, running Map Reduce jobs on the underlying HDFS
- **Sub-Component Performance:** These systems are made up of multiple components, and it is essential to test each of these components in isolation. For example, how quickly the message is indexed and consumed, MapReduce jobs, query performance, search, etc.

- **Performance Testing Approach**
- Performance testing for big data application involves testing of huge volumes of structured and unstructured data, and it requires a specific testing approach to test such massive data.



Performance Testing Approach

- Performance Testing is executed in this order

1. The process begins with the setting of the Big data cluster which is to be tested for performance
2. Identify and design corresponding workloads
3. Prepare individual clients (Custom Scripts are created)
4. Execute the test and analyzes the result (If objectives are not met then tune the component and re-execute)
5. Optimum Configuration