Software Engineering | Requirements Engineering Process

Requirements engineering is the process of identifying, eliciting, analyzing, specifying, validating, and managing the needs and expectations of stakeholders for a software system. The requirements engineering process is an iterative process that involves several steps, including:

- Requirements Elicitation: This is the process of gathering information about the needs and expectations of stakeholders for the software system. This step involves interviews, surveys, focus groups, and other techniques to gather information from stakeholders.
- Requirements Analysis: This step involves analyzing the information gathered in the requirements elicitation step to identify the high-level goals and objectives of the software system. It also involves identifying any constraints or limitations that may affect the development of the software system.
- Requirements Specification: This step involves documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner. This step also involves prioritizing and grouping the requirements into manageable chunks.
- Requirements Validation: This step involves checking that the requirements are complete, consistent, and accurate. It also involves checking that the requirements are testable and that they meet the needs and expectations of stakeholders.
- Requirements Management: This step involves managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant.
- The Requirements Engineering process is a critical step in the software development life cycle as it helps to ensure that the software system being developed meets the needs and expectations of stakeholders, and that it is developed on time, within budget, and to the required quality.

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. it is the disciplined application of proven principle , methods ,tools and notations to describe a proposed system’s intended behavior and its associated constraints.

Tools involved in requirement engineering:

- observation report
- Questionnaire (survey , poll)
- Use cases
- User stories
- Requirement workshop
- Mind mapping
- Role playing
- Prototyping

Requirements Engineering Process consists of the following main activities:

- Requirements elicitation
- Requirements specification
- Requirements verification and validation
- Requirements management

Requirements Elicitation:

It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project. The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Some of these are discussed here. Elicitation does not produce formal models of the requirements understood. Instead, it widens the domain knowledge of the analyst and thus helps in providing input to the next stage.

Requirements elicitation is the process of gathering information about the needs and expectations of stakeholders for a software system. This is the first step in the requirements engineering process and it is critical to the success of the software development project. The goal of this step is to understand the problem that the software system is intended to solve, and the needs and expectations of the stakeholders who will use the system.

There are several techniques that can be used to elicit requirements, including:

- **Interviews**: These are one-on-one conversations with stakeholders to gather information about their needs and expectations.
- **Surveys**: These are questionnaires that are distributed to stakeholders to gather information about their needs and expectations.
- **Focus Groups**: These are small groups of stakeholders who are brought together to discuss their needs and expectations for the software system.
- **Observation**: This technique involves observing the stakeholders in their work environment to gather information about their needs and expectations.
- **Prototyping**: This technique involves creating a working model of the software system, which can be used to gather feedback from stakeholders and to validate requirements.

It's important to document, organize and prioritize the requirements obtained from all these techniques to ensure that they are complete, consistent and accurate.

Requirements specification:

This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process. The models used at this stage include ER diagrams, data flow diagrams(DFDs), function decomposition diagrams(FDDs), data dictionaries, etc.

Requirements specification is the process of documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner. This step also involves prioritizing and grouping the requirements into manageable chunks.

The goal of this step is to create a clear and comprehensive document that describes the requirements for the software system. This document should be understandable by both the development team and the stakeholders.

There are several types of requirements that are commonly specified in this step, including:

- **Functional Requirements:** These describe what the software system should do. They specify the functionality that the system must provide, such as input validation, data storage, and user interface.

- **Non-Functional Requirements**: These describe how well the software system should do it. They specify the quality attributes of the system, such as performance, reliability, usability, and security.
- **Constraints:** These describe any limitations or restrictions that must be considered when developing the software system.
- **Acceptance Criteria**: These describe the conditions that must be met for the software system to be considered complete and ready for release.

In order to make the requirements specification clear, the requirements should be written in a natural language and use simple terms, avoiding technical jargon, and using a consistent format throughout the document. It is also important to use diagrams, models, and other visual aids to help communicate the requirements effectively.

Once the requirements are specified, they must be reviewed and validated by the stakeholders and development team to ensure that they are complete, consistent, and accurate.

Requirements verification and validation:
Verification: It refers to the set of tasks that ensures that the software correctly implements a specific function.
Validation: It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework. The main steps for this process include:

- The requirements should be consistent with all the other requirements i.e no two requirements should conflict with each other.
- The requirements should be complete in every sense.
- The requirements should be practically achievable.

Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

Requirements verification and validation (V&V) is the process of checking that the requirements for a software system are complete, consistent, and accurate, and that they meet the needs and expectations of the stakeholders. The goal of V&V is to ensure that the software system being developed meets the requirements and that it is developed on time, within budget, and to the required quality.

- Verification is the process of checking that the requirements are complete, consistent, and accurate. It involves reviewing the requirements to ensure that they are clear, testable, and free of errors and inconsistencies. This can include reviewing the requirements document, models, and diagrams, and holding meetings and walkthroughs with stakeholders.
- Validation is the process of checking that the requirements meet the needs and expectations of the stakeholders. It involves testing the requirements to ensure that they are valid and that the software system being developed will meet the needs of the stakeholders. This can include testing the software system through simulation, testing with prototypes, and testing with the final version of the software.
- V&V is an iterative process that occurs throughout the software development life cycle. It is important to involve stakeholders and the development team in the V&V process to ensure that the requirements are thoroughly reviewed and tested.

It’s important to note that V&V is not a one-time process, but it should be integrated and continue throughout the software development process and even in the maintenance stage.

Requirements management:

Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible so as to incorporate changes in requirements specified by the end users at later stages too. Being able to modify the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

Requirements management is the process of managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant. The goal of requirements management is to ensure that the software system being developed meets the needs and expectations of the stakeholders and that it is developed on time, within budget, and to the required quality.

There are several key activities that are involved in requirements management, including:

- **Tracking and controlling changes:** This involves monitoring and controlling changes to the requirements throughout the development process, including identifying the source of the change, assessing the impact of the change, and approving or rejecting the change.
- **Version control**: This involves keeping track of different versions of the requirements document and other related artifacts.
- **Traceability**: This involves linking the requirements to other elements of the development process, such as design, testing, and validation.
- **Communication:** This involves ensuring that the requirements are communicated effectively to all stakeholders and that any changes or issues are addressed in a timely manner.
- **Monitoring and reporting**: This involves monitoring the progress of the development process and reporting on the status of the requirements.

Requirements management is a critical step in the software development life cycle as it helps to ensure that the software system being developed meets the needs and expectations of stakeholders, and that it is developed on time, within budget, and to the required quality. It also helps to prevent scope creep and to ensure that the requirements are aligned with the project goals.

ADVANTAGES OF DISADVANTAGES:

The advantages of disadvantages of the requirements engineering process in software engineering include:

Advantages:

- Helps ensure that the software being developed meets the needs and expectations of the stakeholders
- Can help identify potential issues or problems early in the development process, allowing for adjustments to be made before significant
- Helps ensure that the software is developed in a cost-effective and efficient manner

- Can improve communication and collaboration between the development team and stakeholders
- Helps to ensure that the software system meets the needs of all stakeholders.
- Provides a clear and unambiguous description of the requirements, which helps to reduce misunderstandings and errors.
- Helps to identify potential conflicts and contradictions in the requirements, which can be resolved before the software development process begins.
- Helps to ensure that the software system is delivered on time, within budget, and to the required quality standards.
- Provides a solid foundation for the development process, which helps to reduce the risk of failure.

Disadvantages:

- Can be time-consuming and costly, particularly if the requirements gathering process is not well-managed
- Can be difficult to ensure that all stakeholders' needs and expectations are taken into account
- Can be challenging to ensure that the requirements are clear, consistent, and complete
- Changes in requirements can lead to delays and increased costs in the development process.
- As a best practice, Requirements engineering should be flexible, adaptable, and should be aligned with the overall project goals.
- It can be time-consuming and expensive, especially if the requirements are complex.
- It can be difficult to elicit requirements from stakeholders who have different needs and priorities.
- Requirements may change over time, which can result in delays and additional costs.
- There may be conflicts between stakeholders, which can be difficult to resolve.
- It may be challenging to ensure that all stakeholders understand and agree on the requirements.

Requirements engineering is a critical process in software engineering that involves identifying, analyzing, documenting, and managing the requirements of a software system. The requirements engineering process consists of the following stages:

1. Elicitation: In this stage, the requirements are gathered from various stakeholders such as customers, users, and domain experts. The aim is to identify the features and functionalities that the software system should provide.
2. Analysis: In this stage, the requirements are analyzed to determine their feasibility, consistency, and completeness. The aim is to identify any conflicts or contradictions in the requirements and resolve them.
3. Specification: In this stage, the requirements are documented in a clear, concise, and unambiguous manner. The aim is to provide a detailed description of the requirements that can be understood by all stakeholders.
4. Validation: In this stage, the requirements are reviewed and validated to ensure that they meet the needs of all stakeholders. The aim is to ensure that the requirements are accurate, complete, and consistent.

5. Management: In this stage, the requirements are managed throughout the software development lifecycle. The aim is to ensure that any changes or updates to the requirements are properly documented and communicated to all stakeholders.
6. Effective requirements engineering is crucial to the success of software development projects. It helps ensure that the software system meets the needs of all stakeholders and is delivered on time, within budget, and to the required quality standards.

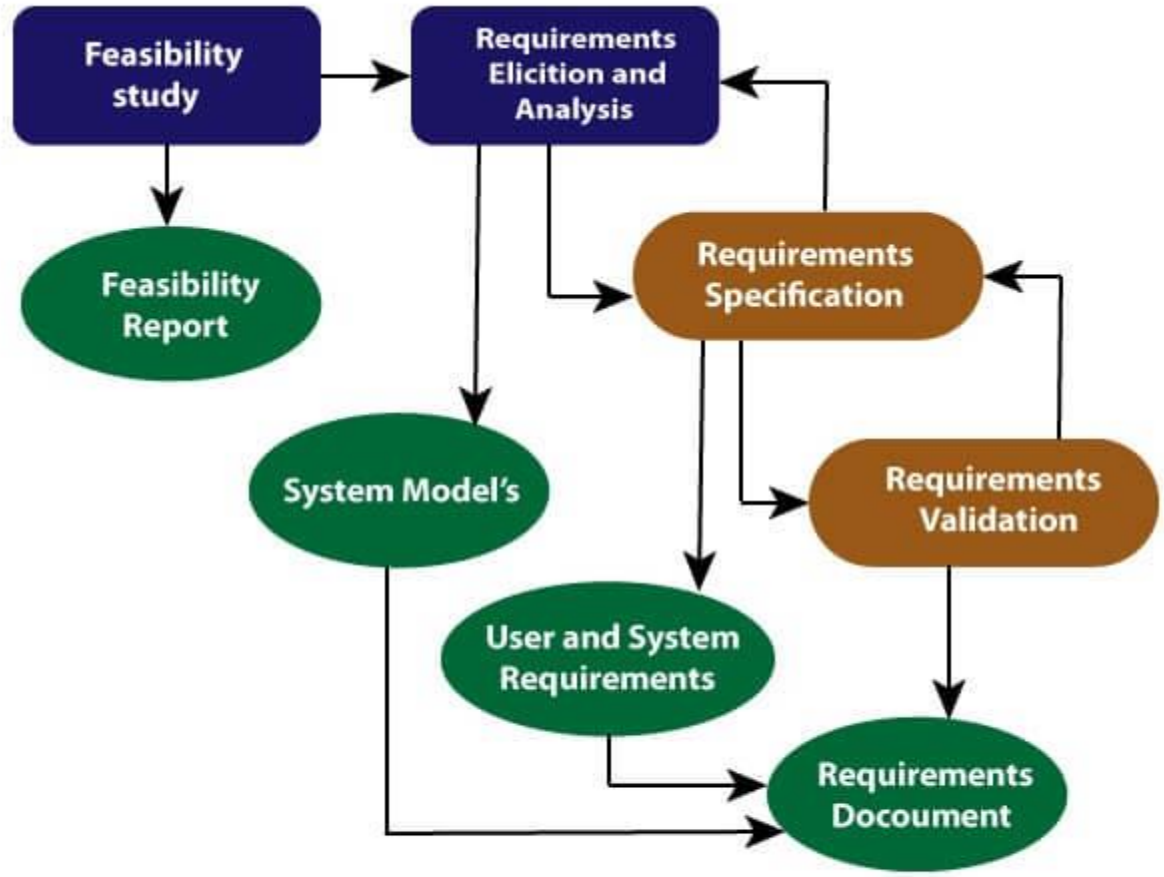
Requirement Engineering

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. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

Requirement Engineering Process

It is a four-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.

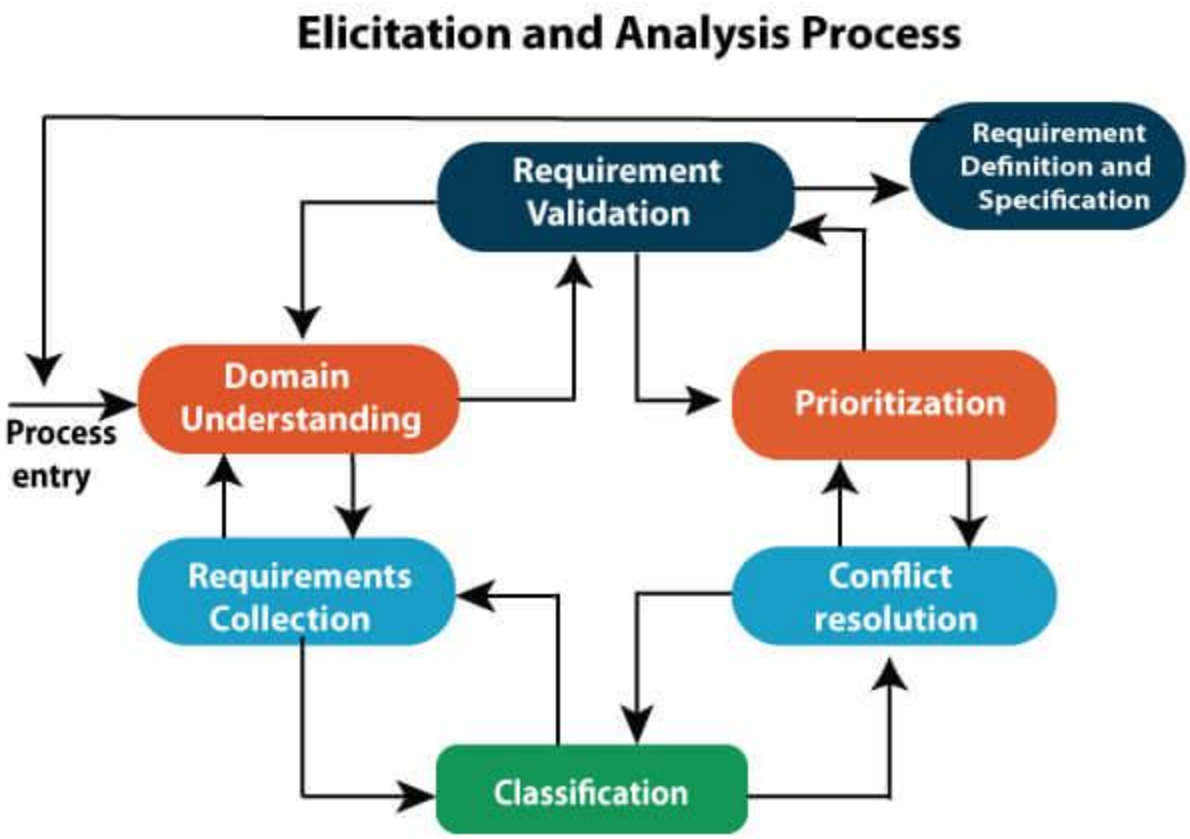
2. Requirement Elicitation and Analysis:

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.



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.

Software Requirement Management:

Requirement management is the process of managing changing requirements during the requirements engineering process and system development.

New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.

The priority of requirements from different viewpoints changes during development process.

The business and technical environment of the system changes during the development.

Prerequisite of Software requirements

Collection of software requirements is the basis of the entire software development project. Hence they should be clear, correct, and well-defined.

A complete Software Requirement Specifications should be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Software Requirements: Largely software requirements must be categorized into two categories:

1. **Functional Requirements:** Functional requirements define a function that a system or system element must be qualified to perform and must be documented in different forms. The functional requirements are describing the behavior of the system as it correlates to the system's functionality.
2. **Non-functional Requirements:** This can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviors of the system. Non-functional requirements are divided into two main categories:
    - **Execution qualities** like security and usability, which are observable at run time.
    - **Evolution qualities** like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. It is about representing a system using

some kind of graphical notation, which is now almost always based on notations in the **Unified Modeling Language** (**UML**). Models help the analyst to understand the functionality of the system; they are used to communicate with customers.

Models can explain the system from **different perspectives**:

- An **external** perspective, where you model the context or environment of the system.
- An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural** perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral** perspective, where you model the dynamic behavior of the system and how it responds to events.

Five types of UML diagrams that are the most useful for system modeling:

- **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.

Models of both new and existing system are used during **requirements engineering**. Models of the **existing systems** help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system. Models of the **new system** are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.

Context and process models

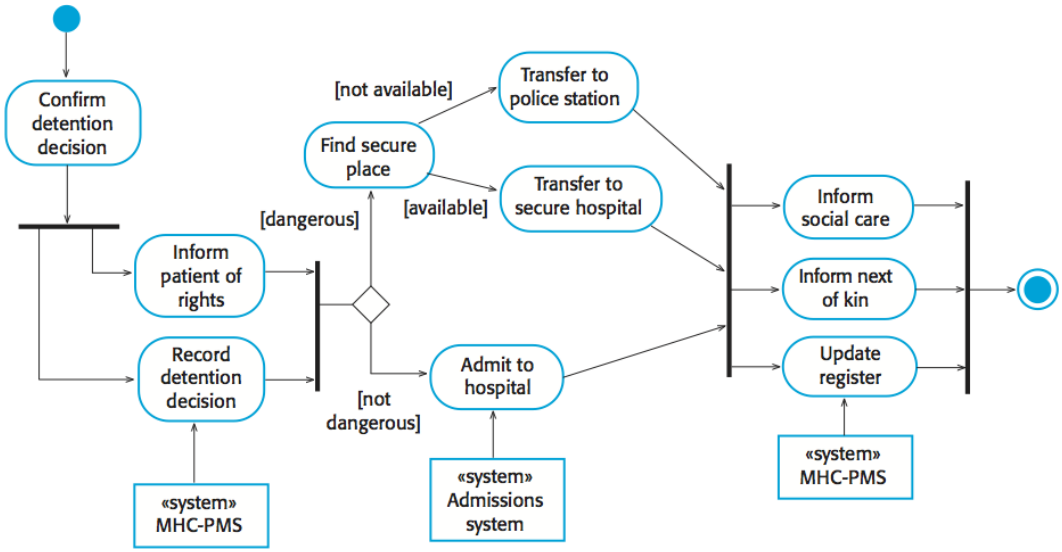
Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries. Social and organizational concerns may affect the decision on where to position system boundaries. Architectural models show the system and its relationship with other systems.

System boundaries 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 since there may be pressures to develop system boundaries that increase/decrease the influence or workload of different parts of an organization.

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

The example below shows a UML **activity diagram** describing the process of involuntary detention and the role of MHC-PMS (mental healthcare patient management system) in it.



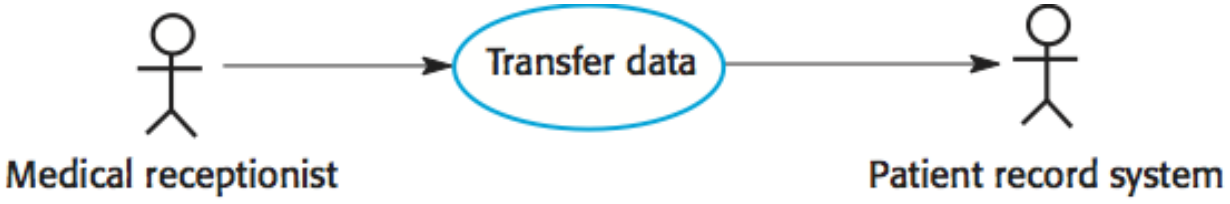
Interaction models

Types of interactions that can be represented in a model:

- Modeling **user interaction** is important as it helps to identify user requirements.
- Modeling **system-to-system interaction** highlights the communication problems that may arise.
- Modeling **component interaction** helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.

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 or other systems. Use cases can be represented using a UML use case diagram and in a more detailed textual/tabular format.

Simple use case diagram:



Use case description in a tabular format:

Use case title	Transfer data

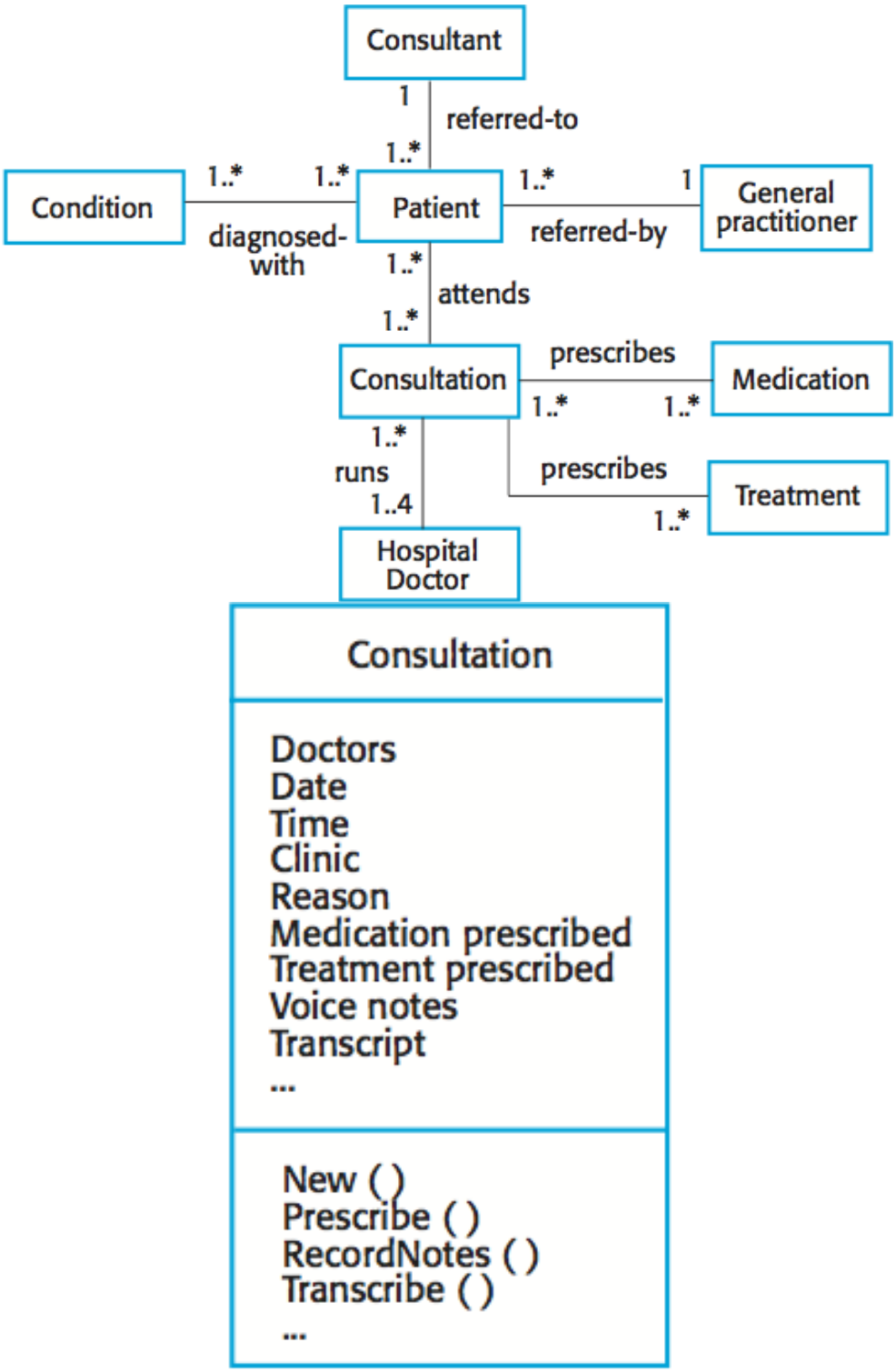
Description	A receptionist may transfer data from the MHC-PMS to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Actor(s)	Medical receptionist, patient records system (PRS)
Preconditions	Patient data has been collected (personal information, treatment summary); The receptionist must have appropriate security permissions to access the patient information and the PRS.
Postconditions	PRS has been updated
Main success scenario	1. Receptionist selects the "Transfer data" option from the menu. 2. PRS verifies the security credentials of the receptionist. 3. Data is transferred. 4. PRS has been updated.
Extensions	2a. The receptionist does not have the necessary security credentials. 2a.1. An error message is displayed. 2a.2. The receptionist backs out of the use case.

UML **sequence diagrams** 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.

Structural models

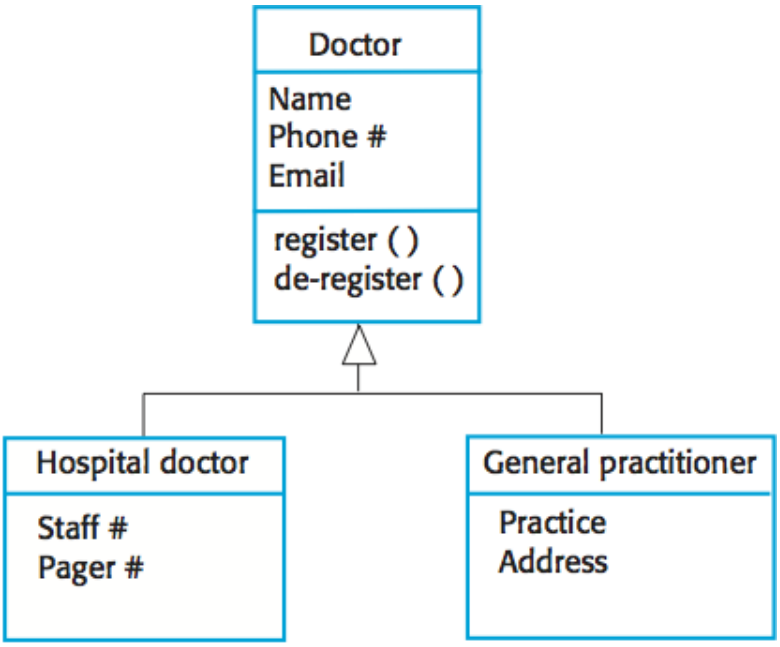
Structural models of software display the organization of a system in terms of the components that make up that system and their relationships. Structural models may be **static** models, which show the structure of the system design, or **dynamic** models, which show the organization of the system when it is executing. You create structural models of a system when you are discussing and designing the system architecture.

UML **class diagrams** are used when developing an object-oriented system model to show the classes in a system and the associations between these classes. An object class can be thought of as a general definition of one kind of system object. An association is a link between classes that indicates that there is some relationship between these classes. When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

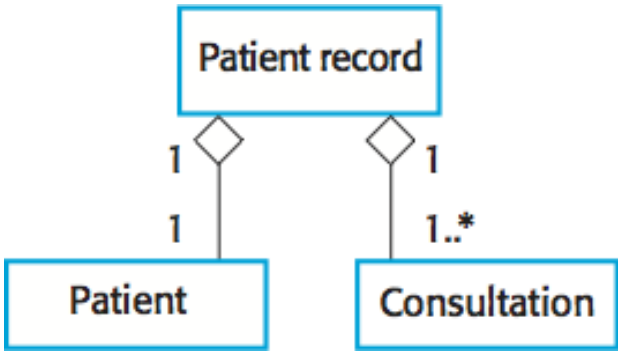


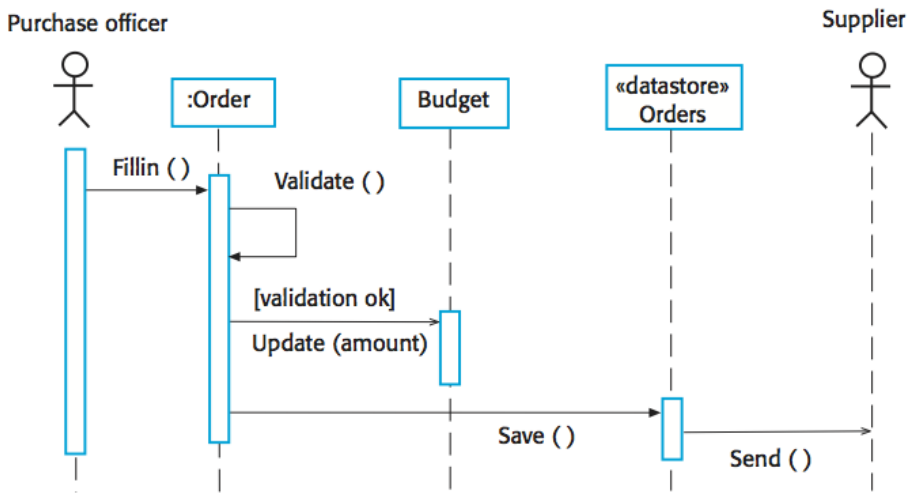
Generalization is an everyday technique that we use to manage complexity. In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. In object-oriented languages, such as Java, generalization is implemented using the class **inheritance** mechanisms built into the language. In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes.

The lower-level classes are subclasses inherit the attributes and operations from their superclasses. These lower-level classes then add more specific attributes and operations.



An **aggregation** model shows how classes that are collections are composed of other classes. Aggregation models are similar to the part-of relationship in semantic data models.





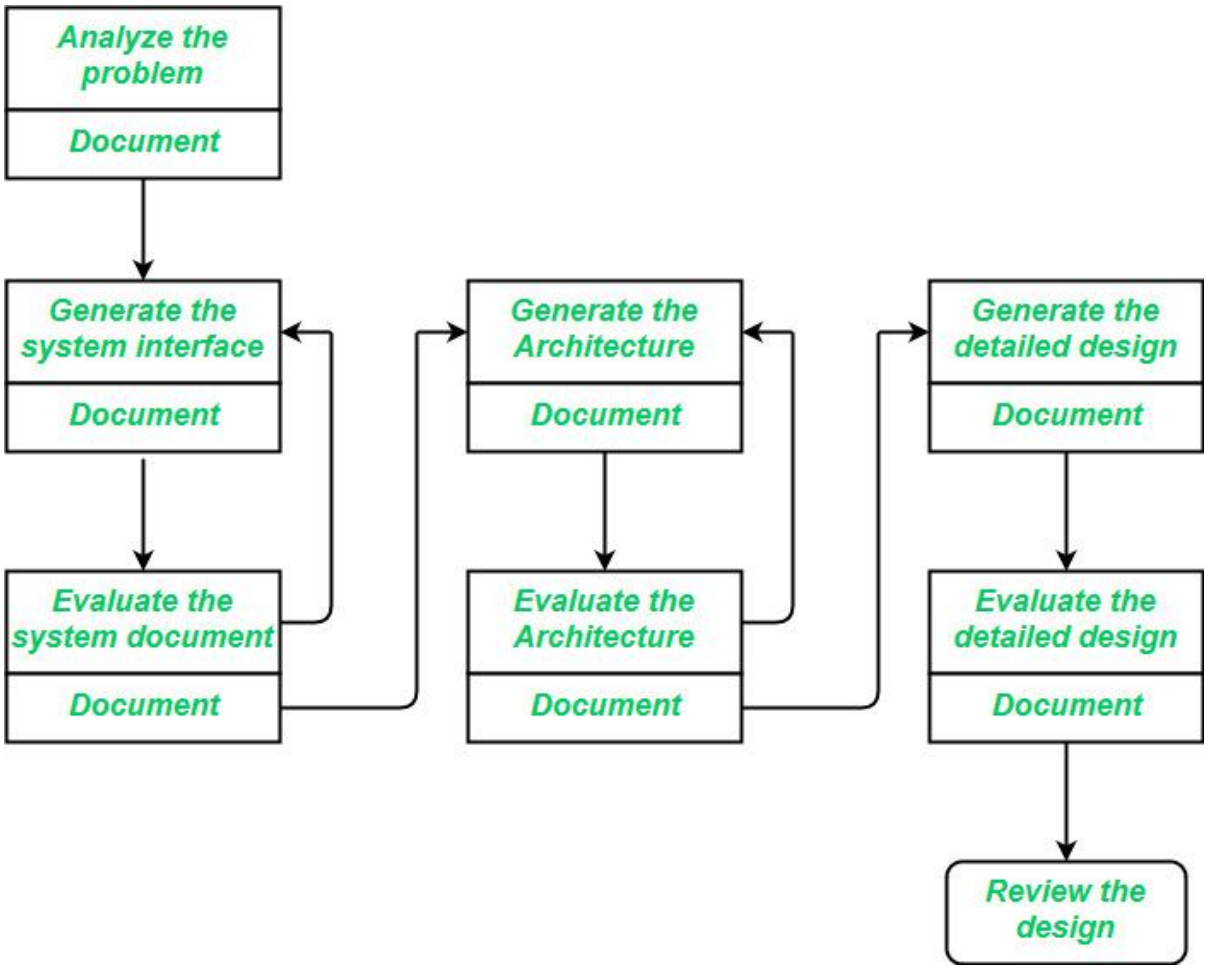
Software Design Process

- The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language. The software design process can be divided into the following three levels of phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design

Elements of a System:

1. **Architecture –** This is the conceptual model that defines the structure, behavior, and views of a system. We can use flowcharts to represent and illustrate the architecture.
2. **Modules –** These are components that handle one specific task in a system. A combination of the modules makes up the system.
3. **Components –** This provides a particular function or group of related functions. They are made up of modules.
4. **Interfaces –** This is the shared boundary across which the components of a system exchange information and relate.
5. **Data –** This is the management of the information and data flow.



Interface Design: *Interface design* is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored and the system is treated as a black box. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called *agents*. Interface design should include the following details:

- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification of the data, and the formats of the data coming into and going out of the system.
- Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

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

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design. **Detailed Design:** *Design* is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures. The detailed design may include:

- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- User interfaces
- Unit states and state changes
- Data and control interaction between units
- Data packaging and implementation, including issues of scope and visibility of program elements
- Algorithms and data structures

Software Quality

- Traditionally, a high-quality product is outlined in terms of its fitness of purpose. That is, a high-quality product will specifically be what the users need to try. For code merchandise, the fitness of purpose is typically taken in terms of satisfaction of the wants arranged down within the SRS document. though “fitness of purpose” could be a satisfactory definition of quality for several merchandise like an automobile, a table fan, a grinding machine, etc. – for code merchandise, “fitness of purpose” isn’t a completely satisfactory definition of quality.
  To convey an associate degree example, think about software that’s functionally correct. It performs all functions as laid out in the SRS document. But, it has an associate degree virtually unusable program. despite the fact that it should be functionally correct, we have a tendency to cannot think about it to be a high-quality product.
  Another example is also that of a product that will have everything that the users need but has an associate degree virtually incomprehensible and not maintainable code. Therefore, the normal construct of quality as “fitness of purpose” for code merchandise isn’t totally satisfactory.

Factors of Software Quality

The modern read of high-quality associates with software many quality factors like the following:

- **Portability:** A software is claimed to be transportable, if it may be simply created to figure in several package environments, in several machines, with alternative code merchandise, etc.
- **Usability:** A software has smart usability if completely different classes of users (i.e. knowledgeable and novice users) will simply invoke the functions of the merchandise.
- **Reusability:** A software has smart reusability if completely different modules of the merchandise will simply be reused to develop new merchandise.
- **Correctness:** Software is correct if completely different needs as laid out in the SRS document are properly enforced.
- **Maintainability:** A software is reparable, if errors may be simply corrected as and once they show up, new functions may be simply added to the merchandise, and therefore the functionalities of the merchandise may be simply changed, etc
- **Reliability.** Software is more reliable if it has fewer failures. Since software engineers do not deliberately plan for their software to fail, reliability depends on the number and type of mistakes they make. Designers can improve reliability by ensuring the software is easy to implement and change, by testing it thoroughly, and also by ensuring that if failures occur, the system can handle them or can recover easily.
- **Efficiency.** The more efficient software is, the less it uses of CPU-time, memory, disk space, network bandwidth, and other resources. This is important to customers in order to reduce their costs of running the software, although with today's powerful computers, CPU time, memory and disk usage are less of a concern than in years gone by.

Software Quality Management System

Software Quality Management System contains the methods that are used by the authorities to develop products having the desired quality.

Managerial Structure: Quality System is responsible for managing the structure as a whole. Every Organization has a managerial structure.

Individual Responsibilities: Each individual present in the organization must have some responsibilities that should be reviewed by the top management and each individual present in the system must take this seriously.

Quality System Activities: The activities which each quality system must have been

- Project Auditing
- Review of the quality system
- It helps in the development of methods and guidelines

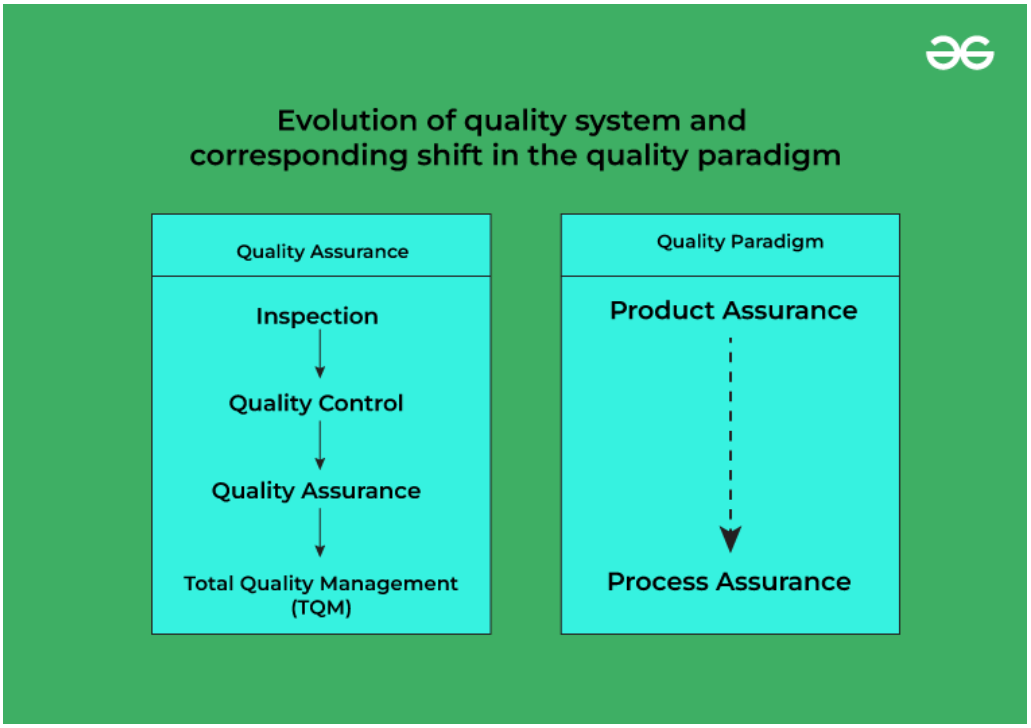
Evolution of Quality Management System

Quality Systems are basically evolved over the past some years. The evolution of a Quality Management System is a four-step process.

The main task of quality control is to detect defective devices and it also helps in finding the cause that leads to the defect. It also helps in the correction of bugs.

Quality Assurance helps an organization in making good quality products. It also helps in improving the quality of the product by passing the products through security checks.

Total Quality Management(TQM) checks and assures that all the procedures must be continuously improved regularly through process measurements.



Software Design process | Set 2

- Software Design is the process to transform the user requirements into some suitable form, which helps the programmer in software coding and implementation. During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document.
  The following items are designed and documented during the design phase:

  - Different modules required.
  - Control relationships among modules.
  - Interface among different modules.
  - Data structure among the different modules.

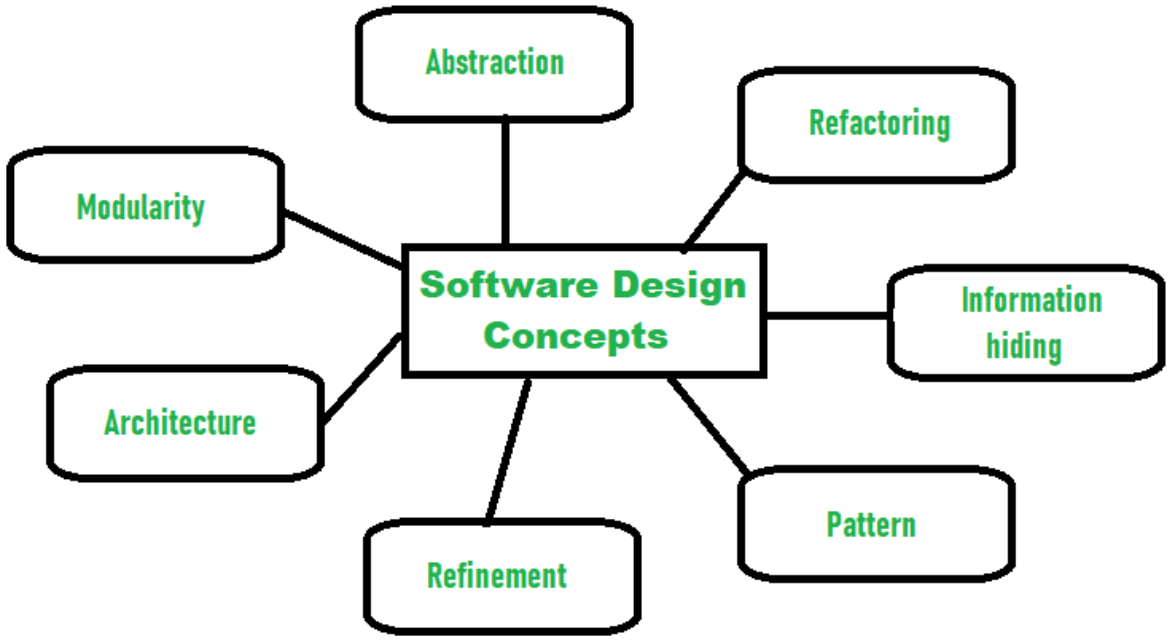
- Algorithms required to implement among the individual modules.

Objectives of Software Design:

1. **Correctness:**
   A good design should be correct i.e. it should correctly implement all the functionalities of the system.
2. **Efficiency:**
   A good software design should address the resources, time, and cost optimization issues.
3. **Flexibility:**
   A good software design should have the ability to adapt and accommodate changes easily. It includes designing the software in a way, that allows for modifications, enhancements, and scalability without requiring significant rework or causing major disruptions to the existing functionality.
4. **Understandability:**
   A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.
5. **Completeness:**
   The design should have all the components like data structures, modules, and external interfaces, etc.
6. **Maintainability:**
   A good software design aims to create a system that is easy to understand, modify, and maintain over time. This involves using modular and well-structured design principles eg.(employing appropriate naming conventions and providing clear documentation). Maintainability in software Design also enables developers to fix bugs, enhance features, and adapt the software to changing requirements without excessive effort or introducing new issues.

Software Design Concepts:

Concepts are defined as a principal idea or invention that comes into our mind or in thought to understand something. The **software design concept** simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software. It allows the software engineer to create the model of the system or software or product that is to be developed or built. The software design concept provides a supporting and essential structure or model for developing the right software. There are many concepts of software design and some of them are given below:



The following **points should be considered while designing Software:**

1. **Abstraction- hide Irrelevant data**
   Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution. The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.
2. **Modularity- subdivide the system**
   Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a trend and is also important. If the system contains fewer components then it would mean the system is complex which requires a lot of effort (cost) but if we are able to divide the system into components then the cost would be small.
3. **Architecture- design a structure of something**
   Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.
4. **Refinement- removes impurities**
   Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is actually a process of

developing or presenting the software or system in a detailed manner that means to elaborate a system or software. Refinement is very necessary to find out any error if present and then to reduce it.

5. **Pattern- a repeated form**
   The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.
6. **Information Hiding- hide the information**
   Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and can't be accessed by any other modules.
7. **Refactoring- reconstruct something**
   Refactoring simply means reconstructing something in such a way that it does not affect the behavior of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without affecting the behavior or its functions. Fowler has defined refactoring as "the process of changing a software system in a way that it won't affect the behavior of the design and improves the internal structure".

Different levels of Software Design:

There are three different levels of software design. They are:

1. **Architectural Design:**
   The architecture of a system can be viewed as the overall structure of the system & the way in which structure provides conceptual integrity of the system. The architectural design identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of the proposed solution domain.

2. **Preliminary or high-level design:**
   Here the problem is decomposed into a set of modules, the control relationship among various modules identified, and also the interfaces among various modules are identified. The outcome of this stage is called the program architecture. Design representation techniques used in this stage are structure chart and UML.

3. **Detailed design:**
   Once the high-level design is complete, a detailed design is undertaken. In detailed design, each module is examined carefully to design the data structure and algorithms. The stage outcome is documented in the form of a module specification document.

Architectural Design

-

Introduction: The software needs the architectural design to represents the design of software. IEEE defines architectural design as "the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system." The software that is built for computer-based systems can exhibit one of these many architectural styles.
Each style will describe a system category that consists of :

- A set of components(eg: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.
- Semantic models that help the designer to understand the overall properties of the system.

The use of architectural styles is to establish a structure for all the components of the system.

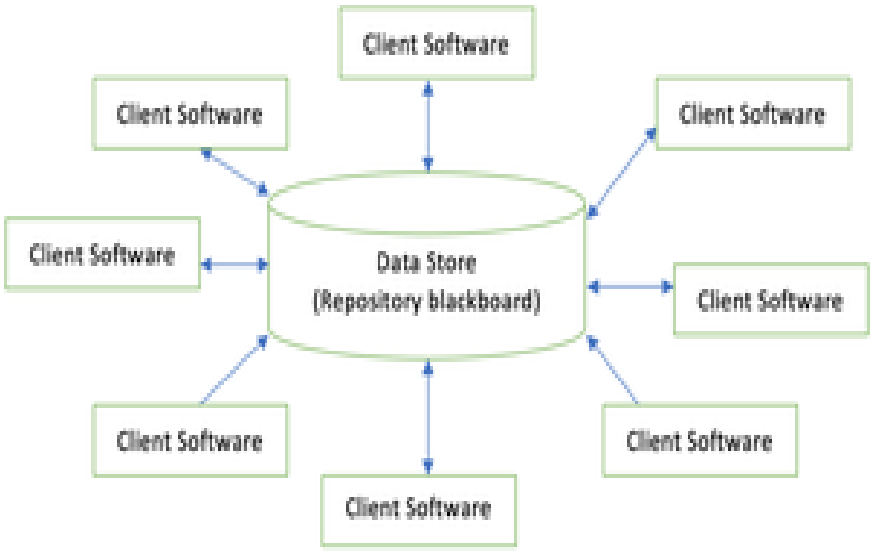
Taxonomy of Architectural styles:

1] Data centered architectures:

- A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
- The figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.
- This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.
- Data can be passed among clients using blackboard mechanism.

Advantage of Data centered architecture

- Repository of data is independent of clients
- Client work independent of each other
- It may be simple to add additional clients.
- Modification can be very easy



2] Data flow architectures:

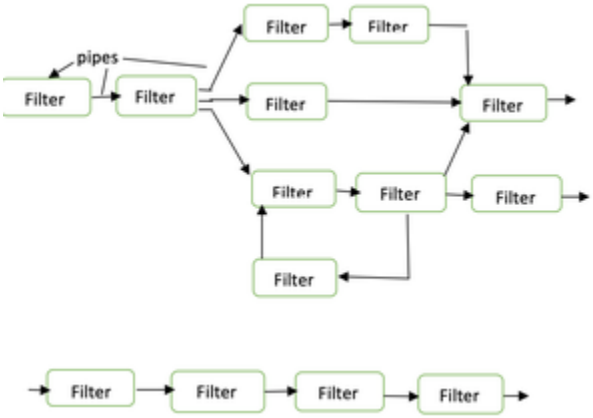
- This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines.
- Pipes are used to transmitting data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don't require any knowledge of the working of neighboring filters.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.

Advantages of Data Flow architecture

- It encourages upkeep, repurposing, and modification.
- With this design, concurrent execution is supported.

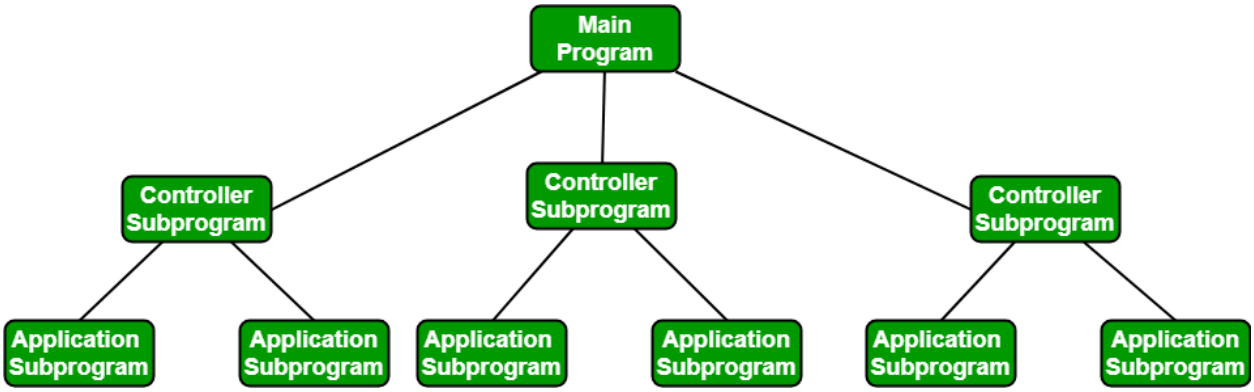
The disadvantage of Data Flow architecture

- It frequently degenerates to batch sequential system
- Data flow architecture does not allow applications that require greater user engagement.
- It is not easy to coordinate two different but related streams



3] Call and Return architectures: It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.

- **Remote procedure call architecture:** This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.
- **Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.



4] Object Oriented architecture: The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

Characteristics of Object Oriented architecture

- Object protect the system's integrity.
- An object is unaware of the depiction of other items.

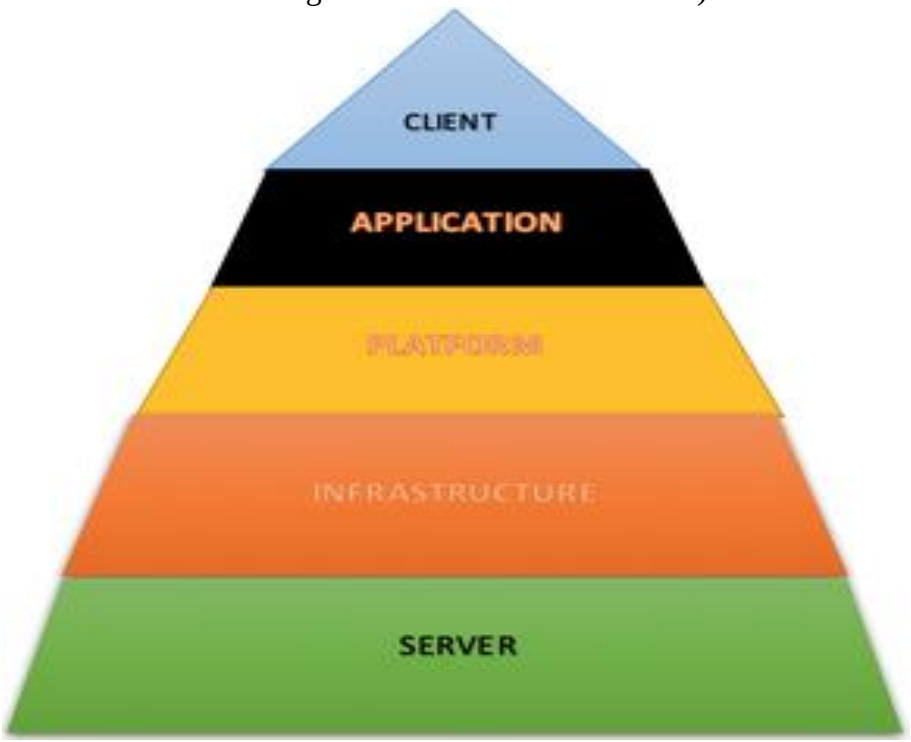
Advantage of Object Oriented architecture

- It enables the designer to separate a challenge into a collection of autonomous objects.

- Other objects are aware of the implementation details of the object, allowing changes to be made without having an impact on other objects.

5] Layered architecture:

- A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
- At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
- Intermediate layers to utility services and application software functions.
- One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International Organisation for Standardisation) communication system.



Layered architecture: