Unit-3
Software Requirements and System Models

- **Software requirements**[1] for a system are the description of what the system should do, the service or services that it provides and the constraints on its operation. The IEEE Standard Glossary of Software Engineering Terminology defines a requirement as:[2]
- A condition or capability needed by a user to solve a problem or achieve an objective
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document
- A documented representation of a condition or capability as in 1 or 2
- The activities related to working with software requirements can broadly be broken down into elicitation, analysis, specification, and management.[3]
- Note that the wording *Software requirements* is additionally used in software release notes to explain, which depending on software packages are required for a certain software to be built/installed/used.

Functional and non-functional requirements

- Requirements analysis is a very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and Non-functional requirements.

Functional Requirements

- These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract.
- These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
- **Example:**
- What are the features that we need to design for this system?
- What are the edge cases we need to consider, if any, in our design?

Non-Functional Requirements

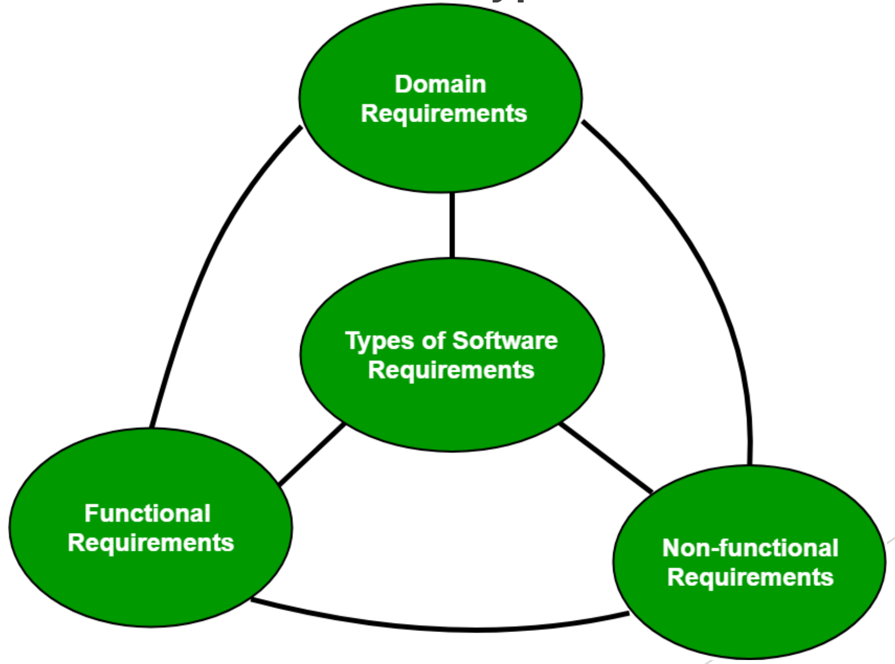
- These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements. They deal with issues like:
- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility
- **Example:**
- Each request should be processed with the minimum latency?
- System should be highly valuable.

Difference between Functional Requirements and Non-Functional Requirements:

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.

User requirements

- **What is Software Requirements?**
- According to IEEE standard 729, a requirement is defined as follows:
- A condition or capability needed by a user to solve a problem or achieve an objective
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
- A documented representation of a condition or capability, as in 1 and 2.
- **Types of Software Requirements**
- **Software Requirements are mainly classified into three types:**



- **1. Functional Requirements**
- **Definition:** Functional requirements describe what the software should do. They define the functions or features that the system must have.
- **Examples:**
- **User Authentication:** The system must allow users to log in using a username and password.
- **Search Functionality:** The software should enable users to search for products by name or category.
- **Report Generation:** The system should be able to generate sales reports for a specified date range.
- **Explanation:** Functional requirements specify the actions that the software needs to perform. These are the basic features and functionalities that users expect from the software.
- **2. Non-functional Requirements**
- **Definition:** Non-functional requirements describe how the software performs a task rather than what it should do. They define the quality attributes, performance criteria, and constraints.
- **Examples:**
- **Performance:** The system should process 1,000 transactions per second.
- **Usability:** The software should be easy to use and have a user-friendly interface.
- **Reliability:** The system must have 99.9% uptime.
- **Security:** Data must be encrypted during transmission and storage.
- **Explanation:** Non-functional requirements are about the system’s behavior, quality, and constraints. They ensure that the software meets certain standards of performance, usability, reliability, and security.

- **3. Domain Requirements**
- **Definition**: Domain requirements are specific to the domain or industry in which the software operates. They include terminology, rules, and standards relevant to that particular domain.
- **Examples**:
- **Healthcare**: The software must comply with HIPAA regulations for handling patient data.
- **Finance**: The system should adhere to GAAP standards for financial reporting.
- **E-commerce**: The software should support various payment gateways like PayPal, Stripe, and credit cards.
- User requirements are statements in **natural language** along with corresponding **diagrams** (tables, forms, intuitive diagrams) detailing the services provided by the system and operational constraints it must comply with. Additionally, it’s worth noting that user requirements primarily focus on the **user’s needs**. Thus, these user requirements cater to the customer.

System requirements

- In software engineering, system requirements play a pivotal role in ensuring that the end product meets the needs of users and stakeholders. These requirements are a comprehensive description of the behavior and capabilities of a system, including both functional and non-functional aspects.
- **Defining System Requirements**
- System requirements are typically divided into two main categories: **user requirements** and **system requirements**. User requirements are statements in natural language, often accompanied by diagrams, that describe the services provided by the system and the operational constraints it must adhere to. These requirements focus on the user's needs and are understandable even to those without technical expertise. They are documented in a "User Requirements Document" using narrative text[1].
- System requirements, on the other hand, are more detailed and technical. They are a structured document that outlines the system's functions, services, and operational constraints. Written primarily for developers, system requirements describe the functionality needed to fulfill user requirements. They act as a blueprint for developers, defining the components that need to be implemented. These requirements are often described in natural language but may also include structured forms and graphical notations. They are documented as a System Requirement Specification (SRS)[1].

- **Types of System Requirements**
- System requirements can be further classified into three types: **functional**, **non-functional**, and **domain requirements**[2].
- **Functional Requirements**: These specify the basic functions that a system must perform. They are the capabilities that the system must have in terms of tasks and services. For example, in a hospital management system, a functional requirement might be the ability for doctors to access patient information.
- **Non-functional Requirements**: These define the quality attributes or constraints the system must meet, such as performance, security, reliability, and usability. They are not directly related to specific behaviors of the system but describe how the system should perform under certain conditions.
- **Domain Requirements**: These are specific to a particular domain or industry and are characteristic of a certain category of projects. They can be either functional or non-functional and are often based on standards or widely accepted feature sets within that domain.
- **Importance of System Requirements**
- The process of defining system requirements is known as **requirements engineering.** It is a critical phase in the software development lifecycle where developers and stakeholders agree on the services and constraints of the system. This phase produces a set of system requirements that serve as the foundation for the contract and guide the development process[1].
- Properly documented system requirements ensure that all parties have a clear understanding of what the system is expected to do. They help in managing expectations and serve as a reference point throughout the development process. Moreover, they facilitate communication among the development team and stakeholders, providing a common language for discussing the system's capabilities and limitations.

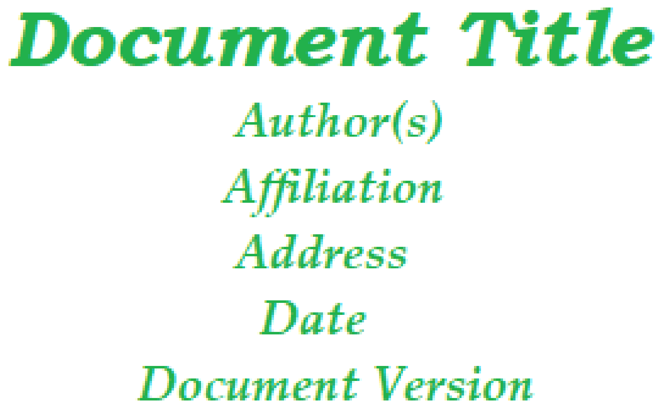
interface specification

- User-interface (UI) design is a critical aspect of software development that focuses on the creation of interfaces for software and devices. The goal of UI design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals—what is often referred to as user-centered design.
- **Key Principles of UI Design**
- When designing user interfaces, there are several key principles and best practices to consider:
- **User-Centered Design**: The design should be tailored around the user's needs, abilities, and preferences. This involves understanding the user's tasks, goals, and context of use. The UI should facilitate the user's tasks and goals efficiently and effectively[1].
- **Consistency**: Consistency in UI design helps users learn the system quickly and apply prior knowledge from one part of the application to another. Consistent use of elements such as icons, color schemes, typography, and terminology is crucial[1][2].
- **Simplicity**: The UI should be simple, avoiding unnecessary elements that do not support user tasks. Complex systems should be broken down into manageable sub-tasks, and information should be presented in a way that's easy to understand[2].
- **Feedback**: Users should receive immediate and clear feedback following their actions. This helps users understand the results of their interactions with the UI[2].
- **Accessibility**: The design should be accessible to users with a wide range of abilities, including those with disabilities. This includes considerations for color contrast, text size, and compatibility with assistive technologies[3] .
- **Flexibility**: The UI should accommodate a range of user preferences and abilities, providing ways to customize the interface[1].

- **Best Practices in UI Design**
- In addition to the key principles, there are several best practices that can enhance the quality of a UI design:
- **Understand the Context**: Designers should have a deep understanding of the context in which the UI will be used. This includes the physical environment, the type of device, and the user's tasks and goals[1].
- **Minimize Cognitive Load**: The UI should minimize the amount of information users need to remember. Information should be visible or easily retrievable when needed[1].
- **Error Handling**: The UI should prevent errors as much as possible and provide simple, understandable mechanisms for handling errors when they do occur[2].
- **Aesthetic and Minimalist Design**: The UI should be aesthetically pleasing but also minimalist, displaying only the information and controls necessary for the user to complete their tasks[2].
- **Control and Freedom**: Users should feel in control of the UI, with the ability to undo and redo actions without penalty[2].

Software Requirement Specification (SRS) Format

- In order to form a good SRS, here you will see some points that can be used and should be considered to form a structure of good Software Requirements Specification (SRS). These are below mentioned in the table of contents and are well explained below.
- **Software Requirement Specification (SRS) Format** as the name suggests, is a complete specification and description of requirements of the software that need to be fulfilled for the successful development of the software system. These requirements can be functional as well as non-functional depending upon the type of requirement. The interaction between different customers and contractors is done because it is necessary to fully understand the needs of customers.
-



Document Title

Author(s)

Affiliation

Address

Date

Document Version

- **Introduction**
- **Purpose of this Document -** At first, main aim of why this document is necessary and what's purpose of document is explained and described.
- **Scope of this document -** In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
- **Overview -** In this, description of product is explained. It's simply summary or overall review of product.
- **General description**
- In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.
- **Functional Requirements**
- In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order. Functional requirements specify the expected behavior of the system-which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system. For each functional requirement, detailed description all the data inputs and their source, the units of measure, and the range of valid inputs must be specified.

- **Interface Requirements**
- In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.
- **Performance Requirements**
- In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc. The performance requirements part of an SRS specifies the performance constraints on the software system. All the requirements relating to the performance characteristics of the system must be clearly specified. There are two types of performance requirements: static and dynamic. Static requirements are those that do not impose constraint on the execution characteristics of the system. Dynamic requirements specify constraints on the execution behaviour of the system.
- **Design Constraints**
- In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc. There are a number of factors in the client's environment that may restrict the choices of a designer leading to design constraints such factors include standards that must be followed resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

- **Non-Functional Attributes**
- In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.
- **Preliminary Schedule and Budget**
- In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.
- **Appendices**
- In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.
- **Uses of SRS document**
- Development team require it for developing product according to the need.
- Test plans are generated by testing group based on the describe external behaviour.
- Maintenance and support staff need it to understand what the software product is supposed to do.
- Project manager base their plans and estimates of schedule, effort and resources on it.
- customer rely on it to know that product they can expect.
- As a contract between developer and customer.
- in documentation purpose.

Feasibility studies :

- **Feasibility Study** in Software Engineering is a study to evaluate feasibility of proposed project or system. Feasibility study is one of stage among important four stages of Software Project Management Process. As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view. Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implementation, contribution of project to the organization etc.
- **Types of Feasibility Study :**
- The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.
- **Technical Feasibility**: In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project. This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development. Along with this, feasibility study also analyzes technical skills and capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.
- **Operational Feasibility:** In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment. Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.
- **Economic Feasibility:** In Economic Feasibility study cost and benefit of the project is analyzed. Means under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. After that it is analyzed whether project will be beneficial in terms of finance for organization or not.
- **Legal Feasibility:** In Legal Feasibility study project is analyzed in legality point of view. This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. Overall it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.

- **Schedule Feasibility:** In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how much time teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can’t be completed on time.
- **Cultural and Political Feasibility:** This section assesses how the software project will affect the political environment and organizational culture. This analysis takes into account the organization’s culture and how the suggested changes could be received there, as well as any potential political obstacles or internal opposition to the project. It is essential that cultural and political factors be taken into account in order to execute projects successfully.
- **Market Feasibility:** This refers to evaluating the market’s willingness and ability to accept the suggested software system. Analyzing the target market, understanding consumer wants and assessing possible rivals are all part of this study. It assists in identifying whether the project is in line with market expectations and whether there is a feasible market for the good or service being offered.
- **Resource Feasibility:** This method evaluates if the resources needed to complete the software project successfully are adequate and readily available. Financial, technological and human resources are all taken into account in this study. It guarantees that sufficient hardware, software, trained labor and funding are available to complete the project successfully.

- **Aim of Feasibility Study**
- The overall objective of the organization are covered and contributed by the system or not.
- The implementation of the system be done using current technology or not.
- Can the system be integrated with the other system which are already exist
- **Feasibility Study Process**
- The below steps are carried out during entire feasibility analysis.
- Information assessment: It assesses the original project concept and establishes the main aims and objectives.
- Information collection: It collects the necessary information and data required to evaluate the project's many components.
- Report writing: It produces an in-depth feasibility report that details the analysis and results.
- General information: It gives a summary of the main points discussed in the report on the feasibility study.

- **Need of Feasibility Study**
- Feasibility study is so important stage of Software Project Management Process as after completion of feasibility study it gives a conclusion of whether to go ahead with proposed project as it is practically feasible or to stop proposed project here as it is not right/feasible to develop or to think/analyze about proposed project again.
- Along with this Feasibility study helps in identifying risk factors involved in developing and deploying system and planning for risk analysis also narrows the business alternatives and enhance success rate analyzing different parameters associated with proposed project development.

Requirements elicitation andanalysis

- **Requirements elicitation** is the process of gathering and defining the requirements for a software system. The goal of requirements elicitation is to ensure that the software development process is based on a clear and comprehensive understanding of the customer’s needs and requirements. This article focuses on discussing Requirement Elicitation in detail.

What is Requirement Elicitation

- The process of investigating and learning about a system’s requirements from users, clients, and other stakeholders is known as requirements elicitation. Requirements elicitation in software engineering is perhaps the most difficult, most error-prone, and most communication-intensive software development.
- Requirement Elicitation can be successful only through an effective customer-developer partnership. It is needed to know what the users require.
- Requirements elicitation involves the identification, collection, analysis, and refinement of the requirements for a software system.
- Requirement Elicitation is a critical part of the software development life cycle and is typically performed at the beginning of the project.
- Requirements elicitation involves stakeholders from different areas of the organization, including business owners, end-users, and technical experts.
- The output of the requirements elicitation process is a set of clear, concise, and well-defined requirements that serve as the basis for the design and development of the software system.
- Requirements elicitation is difficult because just questioning users and customers about system needs may not collect all relevant requirements, particularly for safety and dependability.
- Interviews, surveys, user observation, workshops, brainstorming, use cases, role-playing, and prototyping are all methods for eliciting requirements.

Importance of Requirements Elicitation

- **Compliance with Business Objectives:** The process of elicitation guarantees that the software development endeavors are in harmony with the wider company aims and objectives. Comprehending the business context facilitates the development of a solution that adds value for the company.
- **User Satisfaction**: It is easier to create software that fulfills end users' needs and expectations when they are involved in the requirements elicitation process. Higher user pleasure and acceptance of the finished product are the results of this.
- **Time and Money Savings:** Having precise and well-defined specifications aids in preventing miscommunication and rework during the development phase. As a result, there will be cost savings and the project will be completed on time.
- **Compliance and Regulation Requirements**: Requirements elicitation is crucial for projects in regulated industries to guarantee that the software conforms with applicable laws and norms. In industries like healthcare, finance, and aerospace, this is crucial.
- **Traceability and Documentation:** Throughout the software development process, traceability is based on well-documented requirements. Traceability helps with testing, validation, and maintenance by ensuring that every part of the software can be linked to a particular requirement.

Requirements Elicitation Activities

- Requirements elicitation includes the subsequent activities. A few of them are listed below:
- Knowledge of the overall area where the systems are applied.
- The details of the precise customer problem where the system is going to be applied must be understood.
- Interaction of system with external requirements.
- Detailed investigation of user needs.
- Define the constraints for system development.
- **Features of Requirements Elicitation**
- **Stakeholder engagement:** Requirements elicitation involves engaging with stakeholders such as customers, end-users, project sponsors, and subject-matter experts to understand their needs and requirements.
- **Gathering information:** Requirements elicitation involves gathering information about the system to be developed, the business processes it will support, and the end-users who will be using it.
- **Requirement prioritization:** Requirements elicitation involves prioritizing requirements based on their importance to the project's success.
- **Requirements documentation:** Requirements elicitation involves documenting the requirements clearly and concisely so that they can be easily understood and communicated to the development team.
- **Validation and verification:** Requirements elicitation involves validating and verifying the requirements with the stakeholders to ensure they accurately represent their needs and requirements.
- **Iterative process:** Requirements elicitation is an iterative process that involves continuously refining and updating the requirements based on feedback from stakeholders.
- **Communication and collaboration:** Requirements elicitation involves effective communication and collaboration with stakeholders, project team members, and other relevant parties to ensure that the requirements are clearly understood and implemented.
- **Flexibility:** Requirements elicitation requires flexibility to adapt to changing requirements, stakeholder needs, and project constraints.

Requirements validation

- Requirements validation techniques are essential processes used to ensure that software requirements are complete, consistent, and accurately reflect what the customer wants. These techniques help identify and fix issues early in the development process, reducing the risk of costly errors later on. By thoroughly validating requirements, teams can ensure that the final product meets user needs and expectations. This article focuses on discussing the requirement validation technique in detail.
- **Requirements validation** is the process of checking that requirements defined for development, define the system that the customer wants. To check issues related to requirements, we perform requirements validation. We typically use requirements validation to check errors at the initial phase of development as the error may increase excessive rework when detected later in the development process. In the requirements validation process, we perform a different type of test to check the requirements mentioned in the Software Requirements Specification (SRS), these checks include:
- **Completeness checks**
- **Consistency checks**
- **Validity checks**
- **Realism checks**
- **Ambiguity checks**
- **Variability**

Requirement Validation Techniques

- There are several techniques that are used either individually or in conjunction with other techniques to check entire or part of the system:
- Test Case Generation
- Prototyping
- Requirements Reviews
- Automated Consistency Analysis
- Walk-through
- Simulation
- Checklists for Validation

- **1. Test Case Generation**
- The requirement mentioned in the SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design, this usually means that the requirement will be difficult to implement and it should be reconsidered.
- **2. Prototyping**
- In this validation technique the prototype of the system is presented before the end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is mostly used to collect feedback about the requirement of the user.
- **3. Requirements Reviews**
- In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organizations and the client side, the reviewer systematically analyses the document to check errors and ambiguity.
- **4. Automated Consistency Analysis**
- This approach is used for the automatic detection of an error, such as non-determinism, missing cases, a type error, and circular definitions, in requirements specifications. First, the requirement is structured in formal notation then the CASE tool is used to check the in-consistency of the system, The report of all inconsistencies is identified, and corrective actions are taken.

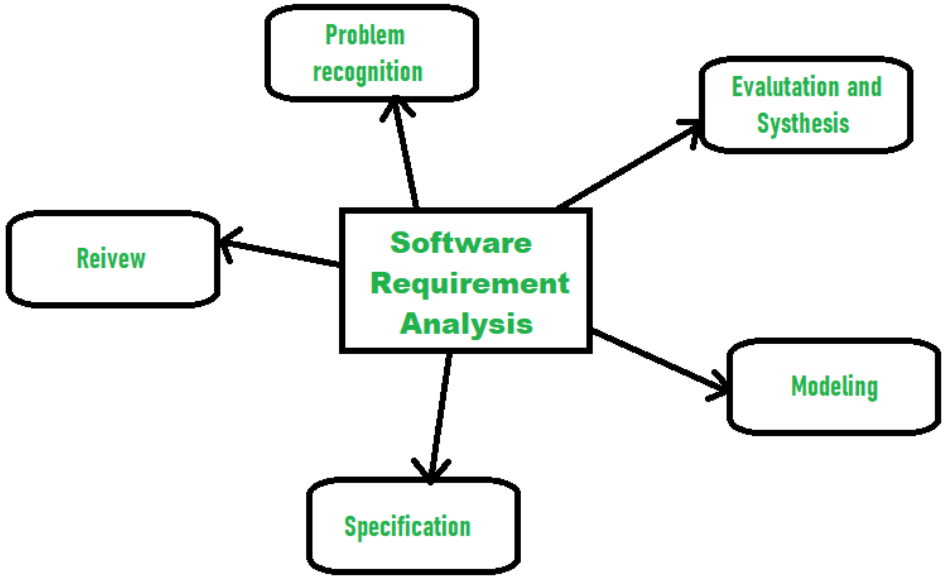
- **5. Walk-through**
- A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.
- Checking early whether the idea is feasible or not.
- Obtaining the opinions and suggestions of other people.
- Checking the approval of others and reaching an agreement.
- **6. Simulation**
- Simulating system behavior in order to verify requirements is known as simulation. This method works especially well for complicated systems when it is possible to replicate real-world settings and make sure the criteria fulfil the desired goals.
- **7. Checklists for Validation**
- It employs pre-made checklists to methodically confirm that every prerequisite satisfies predetermined standards. Aspects like completeness, clarity and viability can all be covered by checklists.

Importance of Requirements Validation Techniques

- **Accuracy and Clarity:** It makes sure that the requirements are precise, unambiguous and clear. This helps to avoid miscommunications and misunderstandings that may result in mistakes and more effort in subsequent phases of the project.
- **User Satisfaction:** It confirms that the requirements meet the wants and expectations of the users, which helps to increase user happiness. This aids in providing a product that satisfies consumer needs and improves user experience as a whole.
- **Early Issue Identification:** It makes it easier to find problems, ambiguities or conflicts in the requirements early on. It is more economical to address these issues early in the development phase rather than later, when the project is far along.
- **Prevents the Scope Creep:** It ensures that the established requirements are well stated and recorded, which helps to prevent scope creep. By establishing defined parameters for the project’s scope, requirements validation helps to lower the possibility of uncontrollably changing course.
- **Improving Quality:** It enhances the software product’s overall quality. By detecting and resolving possible quality problems early in the development life cycle, requirements validation contributes to the creation of a more durable and dependable final product.

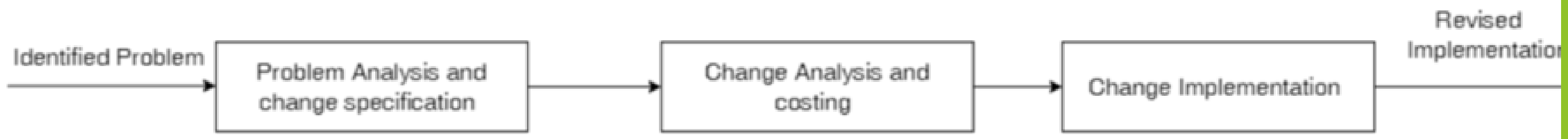
Requirement Management

The requirement management process is the process of managing changing requirements during the requirements engineering process and system development where the new requirements emerge as a system is being developed and after it has gone into use. During this process, one must keep track of individual requirements and maintain links between dependent requirements so that one can assess the impact of requirements changes along with establishing a formal process for making change proposals and linking these to system requirements.



- Now during this phase, there needs to be a certain level of requirement management details which will help to make Requirement Management decisions. To accumulate the details for taking that decision one can follow the following processes:
- **Requirements Identification:** In this, the requirement must be uniquely identified so that it can be cross-referenced with other requirements. Here, one can learn what is important and required and what is not and it also helps to establish a foundation for product vision, scope, cost, and schedule.
- **Requirement change management process:** This is the set of activities that assess the impact and cost of changes.
- **Traceability policies:** The main purpose of this policy is to keep a record of the defined relationships between each requirement and the system designs which will help to minimize the risks.
- **Tool support:** Tools like MS Excel, spreadsheets, or a simple database system can be used.

- Now, after the details have been gathered for the Requirement Management, it's time to see whether the change needs to be implemented or not. For this, we use the *Requirement Change Management process*. In this, the three basic steps that we follow are:
- Problem analysis and change specification
- Change analysis and costing
- Change implementation


Identified Problem
Problem Analysis and change specification
Change Analysis and costing
Change Implementation
Revised Implementatio

Requirement Change Management Process

- At first, the identified problem or the proposal for change is analyzed to ensure its validity. After the analysis of the problem is done, the result is given back to the specific change requestor who may either respond with a more specific requirements change proposal or decide to withdraw the request. Once it is done we have successfully moved to the second phase, where the analysis is done over the effect of the proposed change via traceability policies and general knowledge of the system requirements. Once this analysis is completed, we move to a point where the final decision is to be made on whether or not to proceed with the requirements change.
- If we decide to implement the change then the requirements document and, where necessary, the system design and implementation, are modified. If we decide we do not want to implement the change we eradicate this problem and move to the next. Once the implementation i.e. modification is done as per the request, the implementation is revised and even modified in the document as well so that in the future it can be implemented.
- Finally, in this way, the Requirement Management Process is completed.
- **Advantages of the Requirement Management Process:**
- Recognizing the need for change in the requirements.
- Improved team communication.
- It helps to minimize errors at the early stage of the development cycle.

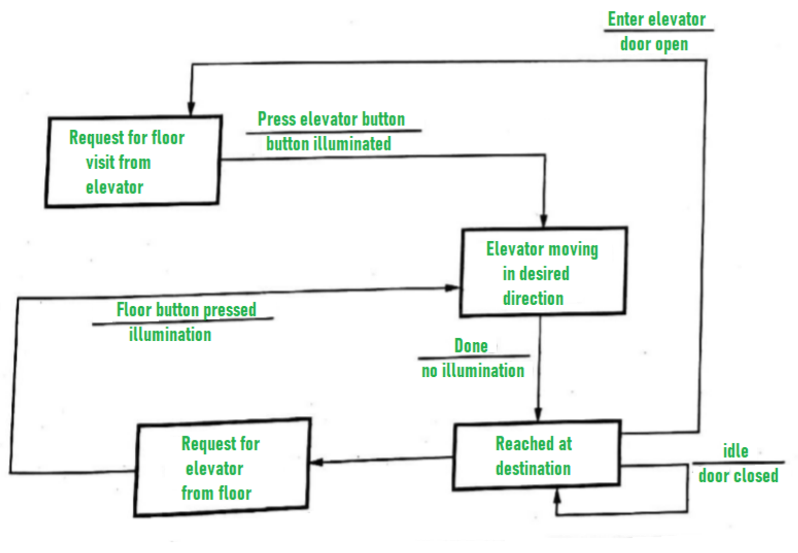
- Context diagrams serve as a foundational tool, helping designers and stakeholders grasp the scope and boundaries of a system under consideration. These diagrams provide a high-level view, illustrating how the system interacts with external entities and the environment. This article explores the significance of context diagrams in system design, their key components, and how to create them.
- **What are Context Diagrams?**
- Context Diagrams are high-level visual representations that show the interactions between a system being developed and its external entities, such as users, other systems, or processes. They provide a big-picture view of how the system fits into its environment without diving into the internal details of the system itself.
- Typically, they consist of a central system surrounded by external entities, with arrows representing data flow or interactions between them.
- They're useful for understanding system boundaries and dependencies.
- **Importance of Context Diagrams in Systems Analysis**
- **Scope Definition:** Context diagrams define the system's boundaries by highlighting its interactions with external entities, ensuring that the analysis focuses on pertinent components and processes.
- **Requirement Gathering:** These diagrams visualize how the system interacts with its environment, aiding in identifying both functional and non-functional requirements. They offer clarity on the system's objectives and its external interactions.
- **Communication:** Acting as a bridge between stakeholders, such as business users, developers, and project managers, context diagrams foster shared understanding of the system's scope and context. They streamline discussions and decision-making throughout the development process.
- **Risk Identification:** Context diagrams assist in spotting potential risks stemming from the system's interactions with external entities. They help stakeholders assess the implications of external factors on the system's performance, security, and reliability.

Behavioral models

- Behavioral models in software engineering are used to represent the dynamic behavior of a system as it interacts with its environment. These models help in understanding how a system responds to various stimuli, which can be either data or events. Here are the main types of behavioral models:
- **Data Processing Model**: This model uses Data Flow Diagrams (DFDs) to represent how data moves through the system. It shows the flow of data between external entities, processes, and data stores. DFDs help in visualizing the data processing steps and the interactions between different components of the system[1].
- **State Machine Model**: This model uses state diagrams to depict the system's response to external events. It shows the different states a system can be in and the transitions between these states triggered by events. State diagrams are particularly useful for modeling systems with complex state-dependent behavior[2].
- Behavioral models are crucial for understanding the overall behavior of a system and ensuring that it meets the required specifications. They provide a clear and visual way to analyze how a system operates and reacts under different conditions[3].

- **Behavioral Model** is specially designed to make us understand behavior and factors that influence behavior of a System. Behavior of a system is explained and represented with the help of a diagram. This diagram is known as State Transition Diagram. It is a collection of states and events. It usually describes overall states that a system can have and events which are responsible for a change in state of a system.
- So, on some occurrence of a particular event, an action is taken and what action needs to be taken is represented by State Transition Diagram.
- **Example :**
  Consider an Elevator. This elevator is for n number of floors and has n number of buttons one for each floor. Elevator's working can be explained as follows :
- **Elevator buttons** are type of set of buttons which is there on elevator. For reaching a particular floor you want to visit, "elevator buttons" for that particular floor is pressed. Pressing, will cause illumination and elevator will start moving towards that particular floor for which you pressed "elevator buttons". As soon as elevator reaches that particular floor,
  illumination gets canceled.
- **Floor buttons** are another type of set of buttons on elevator. If a person is on a particular floor and he wants to go on another floor, then elevator button for that floor is pressed. Then, process will be same as given above. Pressing, will cause illumination and elevator to start moving, and when it reaches on desired floor, illumination gets canceled.

Advantages :
•Behavior and working of a system can easily be understood without any effort.
•Results are more accurate by using this model.
•This model requires less cost for development as cost of resources can be minimal.
•It focuses on behavior of a system rather than theories.
Disadvantages :
•This model does not have any theory, so trainee is not able to fully understand basic principle and major concept of modeling.
•This modeling cannot be fully automated.
•Sometimes, it’s not easy to understand overall result.
•Does not achieve maximum productivity due to some technical issues or any errors.



STATE TRANSITION DIAGRAM

Data models

- **What is Data Modeling?**
- Data modeling is the process of creating a conceptual representation of data and its relationships within a system. It involves defining the structure, constraints, and semantics of data in a way that aligns with the requirements and objectives of the organization or system being developed.
- In simpler terms, data modeling is like creating a blueprint or map that describes how data is organized, stored, and accessed within a system.
- It helps stakeholders, including developers, architects, and business analysts, understand the data requirements, define data entities (such as tables, documents, or objects), specify their attributes, and establish relationships between them.
- **Importance of Data Modeling in System Design**
- **Clarity and Consistency:** Through entities, attributes, and relationships, data structuring and management are brought to a clearer and more consistent level by the process of data modeling in the system.
- **Efficiency:** High-quality data models facilitate information storage and retrieval, and there will be faster performance of the system besides the reduction of resource usage.
- Scalability: The robust data model creates the basis for scalability, by what the systems amounts of the incoming data by means of slowing down their performance or reliability.
- **Data Integrity:** Data modeling offers data accuracy and integrity-checking capabilities by means of data validation, governing the data throughout its existence.
- **Alignment with Business Requirements:** Through the business rules and logic that are embedded in the data model, designers can make sure the system is going well with the business requirements for effectiveness.

Types of Data Models

- **1. Conceptual Data Model**
- It is a high-level, abstract representation of the entities, relationships, and attributes in a system, independent of any specific implementation details.
- Focuses on the business requirements and semantics of the data, providing a clear understanding of the data entities and their relationships.
- Typically used during the initial stages of system design to facilitate communication between stakeholders and guide the development of more detailed data models.
- **2. Logical Data Model**
- It is a detailed representation of the data structures, relationships, and constraints within a system, specifying how data will be organized and stored in a database.
- Translates the concepts defined in the conceptual data model into specific data types, tables, columns, and relationships, often using database-specific constructs such as primary keys, foreign keys, and indexes.
- Enables database designers and developers to design database schemas that are efficient, normalized, and maintainable.

- **3. Physical Data Model**
- It is a concrete representation of the database schema, specifying the physical storage structures, file organization, indexing mechanisms, and other implementation details.
- Maps the logical data model onto the storage mechanisms provided by the underlying database management system (DBMS), taking into account performance considerations, storage constraints, and optimization techniques.
- Guides database administrators in the implementation, configuration, and maintenance of the database system, ensuring optimal performance and scalability.
- **4. Hierarchical Data Model**
- Organizes data in a hierarchical structure, where each data element has a parent-child relationship with other elements, forming a tree-like hierarchy.
- Commonly used in hierarchical databases, where data is organized in parent-child relationships, and each record (node) can have multiple child records.
- Provides fast access to data hierarchies but may be less flexible and scalable compared to other data models.
- **5. Object-Oriented Data Model**
- It represents data using object-oriented concepts such as classes, objects, inheritance, encapsulation,

structured methods

- **Structured Analysis and Structured Design (SA/SD)** is a diagrammatic notation that is designed to help people understand the system. The basic goal of SA/SD is to improve quality and reduce the risk of system failure. It establishes concrete management specifications and documentation. It focuses on the solidity, pliability, and maintainability of the system.
- Structured Analysis and Structured Design (SA/SD) is a software development method that was popular in the 1970s and 1980s. The method is based on the principle of structured programming, which emphasizes the importance of breaking down a software system into smaller, more manageable components.
- In SA/SD, the software development process is divided into two phases: Structured Analysis and Structured Design. During the Structured Analysis phase, the problem to be solved is analyzed and the requirements are gathered. The Structured Design phase involves designing the system to meet the requirements that were gathered in the Structured Analysis phase.
- Structured Analysis and Structured Design (SA/SD) is a traditional software development methodology that was popular in the 1980s and 1990s. It involves a series of techniques for designing and developing software systems in a structured and systematic way. Here are some key concepts of SA/SD:
- Some advantages of SA/SD include its emphasis on structured design and documentation, which can help improve the clarity and maintainability of the system. However, SA/SD has some disadvantages, including its rigidity and inflexibility, which can make it difficult to adapt to changing business requirements or technological trends. Additionally, SA/SD may not be well-suited for complex, dynamic systems, which may require more agile development methodologies.

- SA/SD is combined known as SAD and it mainly focuses on the following 3 points:
- System
- Process
- Technology
- SA/SD involves 2 phases:
- **Analysis Phase:** It uses Data Flow Diagram, Data Dictionary, State Transition diagram and ER diagram.
- **Design Phase:** It uses Structure Chart and Pseudo Code.
- **1. Analysis Phase:**
- Analysis Phase involves data flow diagram, data dictionary, state transition diagram, and entity-relationship diagram.
- **Data Flow Diagram:**
  In the data flow diagram, the model describes how the data flows through the system. We can incorporate the Boolean operators and & or link data flow when more than one data flow may be input or output from a process. For example, if we have to choose between two paths of a process we can add an operator or and if two data flows are necessary for a process we can add an operator. The input of the process "check-order" needs the credit information and order information whereas the output of the process would be a cash-order or a good-credit-order. **Data Dictionary:**
  The content that is not described in the DFD is described in the data dictionary. It defines the data store and relevant meaning. A physical data dictionary for data elements that flow between processes, between entities, and between processes and entities may be included. This would also include descriptions of data elements that flow external to the data stores. A logical data dictionary may also be included for each such data element. All system names, whether they are names of entities, types, relations, attributes, or services, should be entered in the dictionary.
- **State Transition Diagram:**
  State transition diagram is similar to the dynamic model. It specifies how much time the function will take to execute and data access triggered by events. It also describes all of the states that an object can have, the events under which an object changes state, the conditions that must be fulfilled before the transition will occur and the activities were undertaken during the life of an object.
- **ER Diagram:**
  ER diagram specifies the relationship between data store. It is basically used in database design. It basically describes the relationship between different entities.

- **2. Design Phase:**
- Design Phase involves structure chart and pseudocode.
- **Structure Chart:**
  It is created by the data flow diagram. Structure Chart specifies how DFS's processes are grouped into tasks and allocated to the CPU. The structured chart does not show the working and internal structure of the processes or modules and does not show the relationship between data or data flows. Similar to other SASD tools, it is time and cost-independent and there is no error-checking technique associated with this tool. The modules of a structured chart are arranged arbitrarily and any process from a DFD can be chosen as the central transform depending on the analysts' own perception. The structured chart is difficult to amend, verify, maintain, and check for completeness and consistency.
- **Pseudo Code:** It is the actual implementation of the system. It is an informal way of programming that doesn't require any specific programming language or technology.

- **Advantages of Structured Analysis and Structured Design (SA/SD):**
- Clarity and Simplicity: The SA/SD method emphasizes breaking down complex systems into smaller, more manageable components, which makes the system easier to understand and manage.
- Better Communication: The SA/SD method provides a common language and framework for communicating the design of a system, which can improve communication between stakeholders and help ensure that the system meets their needs and expectations.
- Improved maintainability: The SA/SD method provides a clear, organized structure for a system, which can make it easier to maintain and update the system over time.
- Better Testability: The SA/SD method provides a clear definition of the inputs and outputs of a system, which makes it easier to test the system and ensure that it meets its requirements.
- **Disadvantages of Structured Analysis and Structured Design (SA/SD):**
- Time-Consuming: The SA/SD method can be time-consuming, especially for large and complex systems, as it requires a significant amount of documentation and analysis.
- Inflexibility: Once a system has been designed using the SA/SD method, it can be difficult to make changes to the design, as the process is highly structured and documentation-intensive.
- Limited Iteration: The SA/SD method is not well-suited for iterative development, as it is designed to be completed in a single pass.