Software requirements define the capabilities, functions, and constraints of a software system. They serve as a foundation for software development, ensuring that the system meets user needs, business goals, and technical specifications.

A Software Requirement can be described as: A documented statement of what a software system must do, how it should behave, and any constraints that must be met for successful operation.

Importance of software requirements

1. Clear vision and direction

• Defines Scope: Requirements provide a clear understanding of the software's goals and boundaries. This ensures that both the development team and stakeholders (clients, users, managers) are aligned on what the software should achieve.

• Guides Development: They help define the tasks, priorities, and features for developers, ensuring that efforts are focused on delivering the intended product.

2. Ensures stakeholder satisfaction

• Aligns Expectations: Clear requirements help ensure that all stakeholders, including end users, business owners, and development teams, have the same understanding of the project scope and functionality.

• Prevents Miscommunication: They reduce the chances of misinterpretations between users, developers, and other stakeholders by setting out the exact needs and objectives.

3. Project planning and estimation

• Accurate Time and Cost Estimates: With clear requirements, teams can better estimate the time, resources, and cost needed to complete the project, allowing for better budgeting and scheduling.

• Risk Management: Having well-documented requirements helps to identify potential challenges or issues early in the project lifecycle, which can help mitigate risks.

4. Quality assurance

• Basis for Testing: Requirements provide a solid foundation for creating test cases and validation criteria, ensuring the software behaves as expected.

• Ensures Completeness: By having documented requirements, developers and testers can verify that all features and functions are delivered, reducing the risk of missing critical functionalities.

5. Reduces scope creep

• Controlled Changes: Well-defined requirements act as a baseline to manage changes. Without a clear reference, scope creep (uncontrolled changes or additions to the project) can lead to missed deadlines, increased costs, or a product that doesn't meet the original needs.

• Helps with Change Management: Any changes or additions to the requirements can be tracked and analyzed to understand their impact on the timeline and budget.

Types of software requirements

1. Functional Requirements
2. Non-Functional Requirements
3. System Requirements
4. User Requirements

Functional and Non-Functional Requirements

1. Functional requirements

Functional requirements define the specific behavior or functions that a system, application, or product must be able to perform. They focus on the actions the software must take to meet the user needs and business goals.

Functional requirements are often described in terms of inputs, processes, and outputs, detailing what the software should do.

Functional requirements specify what the product must do in order to satisfy the basic reason for its existence. They are

1. Specifications of the product's functionality.
2. Actions that the product must take; check, compute, record, and retrieve.
3. Derived from the basic purpose of the product.
4. Normally business oriented, rather than technical.
5. Not the technical solution constraints that are often referred as the 'system requirements
6. Derived mostly from the use case scenarios.
7. Not a quality.
8. Not measurable or testable at this stage.

2. Non-functional requirements

Non-functional requirements (NFRs) refer to the quality attributes, system characteristics, and constraints that describe how a system should behave, rather than what it should do. They focus on the performance, usability, reliability, scalability, security, and other aspects that impact the overall experience and operational efficiency of the system.

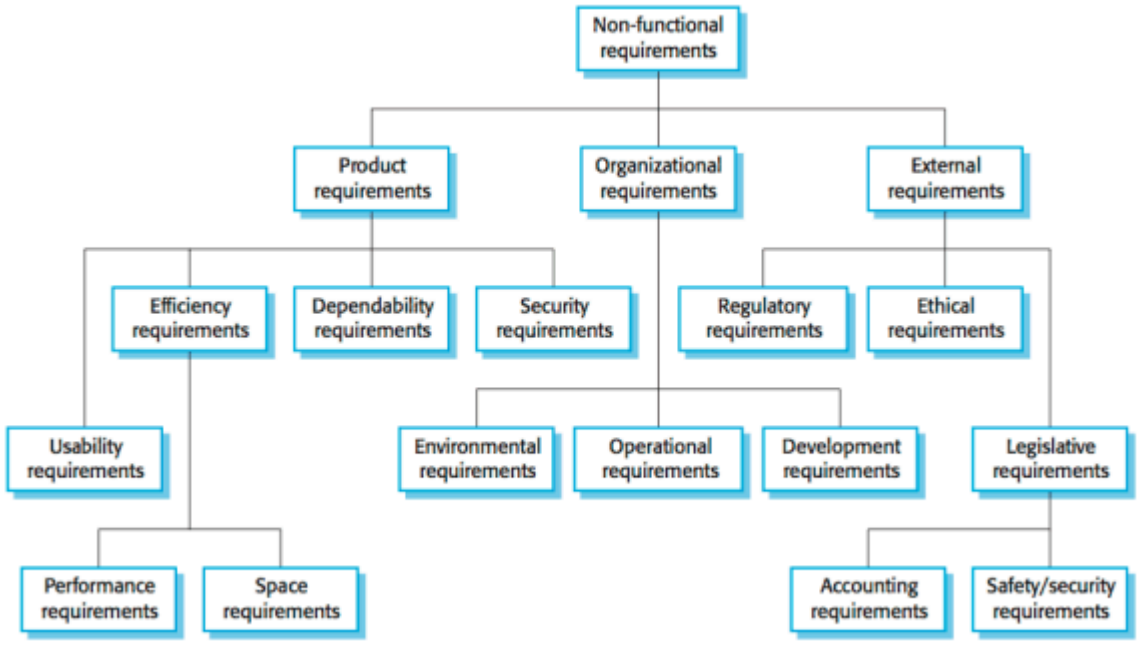
Hence, they are critical than functional requirements. Using these requirements, a specimen can easily avoid the errors occurring due to miss-implemented functional requirements. For example, no one would prefer of purchasing a water heater if its operation is unreliable i.e., if the heater is shock prone.

While dealing with non-functional requirements, one has to remember; it not only deals with the end product or the system (being developed) but also its process of manufacturing.

There are many reasons for the outcome of non-functional requirements.
They are listed below:

• Budget constraints.
• Inadequate user needs.
• If there is requirement of one system to be operable over other software/hardware.
• Strong privacy and safety constraints.
• Strong organizational policies etc

Classification of non-functional requirements



Product requirements

Requirements which specify that the delivered product must behave in a particular way, example: execution speed, reliability etc.

Organizational requirements

Requirements which are a consequence of organizational policies and procedures, example: process standards used, implementation requirements etc.

External requirements

Requirements which arise from factors which are external to the system and its development process, example: Interoperability requirements, legislative requirements etc.

Examples of functional and non-functional requirements

1. Online banking system

Functional requirements:
• Users should be able to log in with their username and password.
• Users should be able to check their account balance.
• Users should receive notifications after making a transaction.

Non-functional requirements:
• The system should respond to user actions in less than 2 seconds.
• All transactions must be encrypted and comply with industry security standards.
• The system should be able to handle 100 million users with minimal downtime.

2. Food delivery app

Functional requirements:
• Users can browse the menu and place an order.
• Users can make payments and track their orders in real time.

Non-functional requirements:
• The app should load the restaurant menu in under 1 second.
• The system should support up to 50,000 concurrent orders during peak hours.
• The app should be easy to use for first-time users, with an intuitive interface.

User Requirements

User requirements include both functional as well as non-functional details. The details should only provide the external behavioral aspects of the system. Hence, it should not include its implementation details which may frustrate the nontechnical users. The details should be written in terms of diagrams, forms, tables etc., to a major extent. Apart from this, there may be certain other problems which may evolve due to the usage of natural language in writing these specifications.

Few of these problems are listed below

Problem 1: Requirements amalgamation

This problem usually arises when several requirements are clubbed together and are specified as one.

Problem 2: Lack of clarity

When the requirements are not specified in a precise and unambiguous way, then clarity is said to be lacking.

Problem 3: Requirements confusion

This problem usually arises whenever the developers do not specify a clear distinction between non-functional requirements, functional requirements, design information and system goals.

Moreover, while writing user requirements, one has to remember that it should not be included along with the details of the system requirements. The user requirements should be precise. If it covers large amount of data then the software developer cannot include solutions to the frequently occurring problems and it becomes difficult to read. Hence, it should provide only the essential facts, while avoiding the unnecessary details. Whenever any such details are provided, a supporting reason (behind its inclusion) should be associated. This reason describes its importance and it can be considered whenever suitable changes are made to the system.

Following are certain important, principle guidelines recommended to be followed while preparing the user requirements

(i) It is a good practice to highlight the useful information.
Highlighting can be done by either making the data italic, underline, bold or by changing its colour.

(ii)The user requirements documents should not get mixed up with the system using the information.

(iii) While writing requirements, it is good practice to follow standard notations throughout the document. For Example the requirements can be in bold format while reason of its inclusion in normal text.

(iv) Further, in order to make the document more realistic, the source from which the given reason is acquired can also be associated with a particular reason. This provides ease in considering the source when suitable changes are being made to these requirements.

(v) While specifying the requirements, there should be clear distinction between the mandatory and desirable requirements.

(vi) Computer jargons must be avoided to the possible extent.

System Requirements

‘System Requirements’ is often the initial step in the system designing process. They provide the implementation details of the user requirements. Hence, they are more detailed than user requirements. The data within the system requirements should be complete as well as consistent.

System requirements should only consider the description of the operational as well as behavioral aspects of the systems. It should not include the implementation as well as the design details of the system. But, whenever large and complex systems are considered the design details are avoided in the system requirements. There are many reasons supporting this occasion. Few of them are quoted below.

If the system is distributed in nature (i.e., the system supporting interoperability), the design of a given system should be focused on the system requirements.

In order to ensure that the current system satisfies nonfunctional requirements, system architecture need to be considered.

However, there many other factors which can create complexities while documenting the system requirements. The fundamental factor is the usage of natural language while documenting these requirements.

They not only make the text difficult to understand, but they are often confusing.

Following is a list of reasons supporting the illustration

(i) As the requirements are specified by a developer in his own notion, it rests with the end user to properly make the distinction between the available requirements from the given volumes of text. This is because, given requirement may be grammatically analogous to other available text or given requirements may look similar but refer to two different facts etc.).

(ii) It is often impractical to associate the requirements under different side headings. Hence, a single requirement will drive through the entire system requirements set.

(iii) If the writer is addicted to using certain or desired phrases on different occasions, then readers may misunderstand the concept. This may deviate them from the actual requirements. Sometimes these facts are analyzed during final phases of the software development. On those occasions the costs of development will eventually rise. When these problems were analyzed by the scientists, they concluded to use a language (while writing the requirements) which can be easily grasped by all the individuals involved.

This provides a full opportunity to the writers to express their views in the most appropriate manner.

System Requirements

'System Requirements' is often the initial step in the system designing process. They provide the implementation details of the user requirements. Hence, they are more detailed than user requirements. The data within the system requirements should be complete as well as consistent.

System requirements should only consider the description of the operational as well as behavioral aspects of the systems. It should not include the implementation as well as the design details of the system. But, whenever large and complex systems are considered the design details are avoided in the system requirements. There are many reasons supporting this occasion. Few of them are quoted below.

If the system is distributed in nature (i.e., the system supporting interoperability), the design of a given system should be focused on the system requirements.

In order to ensure that the current system satisfies nonfunctional requirements, system architecture need to be considered.

However, there many other factors which can create complexities while documenting the system requirements. The fundamental factor is the usage of natural language while documenting these requirements.

They not only make the text difficult to understand, but they are often confusing.

Following is a list of reasons supporting the illustration

(i)As the requirements are specified by a developer in his own notion, it rests with the end user to properly make the distinction between the available requirements from the given volumes of text. This is because, given requirement may be grammatically analogous to other available text or given requirements may look similar but refer to two different facts etc.).

(ii) It is often impractical to associate the requirements under different side headings. Hence, a single requirement will drive through the entire system requirements set.

(iii)If the writer is addicted to using certain or desired phrases on different occasions, then readers may misunderstand the concept. This may deviate them from the actual requirements. Sometimes these facts are analyzed during final phases of the software development. On those occasions the costs of development will eventually rise. When these problems were analyzed by the scientists, they concluded to use a language (while writing the requirements) which can be easily grasped by all the individuals involved**.**

This provides a full opportunity to the writers to express their views in the most appropriate manner.

Using the language, the requirements are framed in a well planned format with suitable models used based on the requirements. The structure can also have models exchanging dialogues and other perspectives.

Following are the set of notations which can be used for the requirement specifications

Specifying system requirements

Notation: Graphical notations

Illustration: This refers to representation consisting of certain graphical objects along with appropriate text inscribed at suitable locations. Good examples of such notations are SADIT, use case diagrams and sequence diagrams.

Notation: Structured natural language

Illustration: This notation facilitates the writers to utilize certain predefined formats as well as templates while expressing a scenario in the form of requirement.

Notation: Mathematical specifications

Illustration: These notations use the mathematical concepts like sets or finite state machines. Even though these notations minimize customer – contractor arguments, customers hesitate to accept it as a system contract.

Notation: Design description language

Illustration: In this case, the writers can utilize certain programming languages, in narrating the given requirements in terms of an operational model, if the model reflects only the abstract notion of the requirement.

Structured natural language specifications

Structured Natural Language was initially proposed by Heninger with an intention of providing certain standards while writing the system requirements for an aircraft software system.

According to him, structured natural language is a decent style of writing system requirements which privileges the writers with well defined standards. It also helps in curbing writers independency of using their own visualizations. Hence, there will be high degree of expressibility but this expressibility can be adhered with equal measures of uniformity.

Structured natural language helps in decreasing the number of words used in describing the context of the system. It allows to look forward for usage of templates whenever necessary. Also focus on some of these requirements (which remains extremely essential), structured natural languages highlight them. Hence, in other words, the structured natural language expresses the system requirements in certain standard forms,

While dealing with these form-based writing of system requirements, following illustrations must be considered

(i)There should be significant description corresponding to a function or a given entity.

(ii) Significant description corresponding to the inputs of the system as well as the source producing them. Also the description of outputs and the entities using them.

(iii) Illustrations related to the actions to be performed.

(iv)Illustration corresponding to generation of side effects while performing a given operation.

(v)Illustration on initial, as well as final conditions existing before and after a given function, if the applications consisting of functions.

Domain analysis

Domain Analysis plays an important role in specifying whether the system can function properly or not. This is because domain requirements focus on the system's application domain and not on the user's and system's requirements. While performing requirement engineering, the analysis pattern is performed repeatedly across several applications with respect to a particular business domain. If analysis pattern can be defined or categorized in a way that helps in identifying and reusing the pattern, then it is possible to create an analysis model. The impact of such a creation is that the use of reusable design pattern and executable software component increases. This intern enhances the time-to-market and decreases the cost associated with the development of the pattern.

Domain Analysis contains the information required for recognizing, defining and categorizing the analysis pattern.

According to Firesmith, software domain analysis is defined as a process of identifying, analyzing and specifying the common reusable requirements from a specific application domain.

Firesmith also defined object oriented domain analysis as a process of identifying, analyzing and specifying the common reusable capabilities present in a specific application domain.

These capabilities are defined in terms of classes, objects and frameworks.

The specific application domain can be of wider range i.e., it basically ranges from avionics to banking or from multimedia video games to software embedded systems. The objective of domain analysis is to identify, create analysis classes, functionality that are mainly applied for making them reusable. The information about expert advice, customer surveys and existing applications are extracted from sources of domain knowledge and are provided as inputs to domain analysis. Once the given inputs are processed, the outputs generated are class taxonomies, reuse standard and domain languages. These outputs are intum supplied as input to domain analysis model. The main purpose of using domain knowledge is to identify reusable objects across the specific application domain.

The entire process of domain analysis is characterized based on the context related to the object oriented software engineering. This process can be applied not only to the software engineering model but also to the traditional object oriented design models.

The following are the steps carried out while performing the domain analysis process

1. The domain that needs to be analyzed must be defined.
2. The items retrieved from sources of domain knowledge must be categorized.
3. The sample of application defined in the domain must be collected.
4. Every individual application in the sample is analyzed. Based on this, analysis class must be defined.
5. An analysis model forth class must be developed.

Interface Specification

Software systems must be able to work with systems that are already implemented and installed. Thus, interfaces between them must be precisely specified. Specifications related to interfaces must be included in the requirements document as early as possible.

Types of interfaces

1. Procedural interfaces

Application Programming Interfaces (API) are the procedural interfaces. Any service offered by a program or sub-system can be accessed by invoking an interface procedure.

2. Data structures

A sub-system may pass data structures to another sub-system. To describe these structures, Java or C++ automatically generated descriptions or graphical data models are used.

3. Representations of data

The way data is represented, for example, bit ordering, can be used as an interface. Most of the real-time embedded systems use these interfaces. Moreover, Ada also supports them.

Software requirements document

1. Both customer as well as system requirements specifications can be integrated to form one document.
   (or)
   Customer requirements specification can be provided in the initial phase or at the introduction of the document. This step is followed by detailed system requirement specification.
   (or)
   If the application is complex and probably large then system requirement specifications can be included at separate locations or document. Usually, in each software requirements document the details of large sets of stakeholders involved in the development are included.

Following is a list of stakeholders and illustrations related to their involvement in the development,

(a) System customer and end user

These are the people to whom software is being delivered. They are authenticated to supply their requirements and also read the current document (software requirement document) in order to ensure that the (document) is built in accordance to their needs.

(b) System managers

System managers usually consider the document to decide on the cost of development and to plan out the consequences of entire project.

(c) Development team or engineers

They usually decide on the tasks to be accomplished.

(d) Testers

They use the documents in order to develop suitable test cases.

(e) Maintenance engineers

They consider the document as a mode to understand as well as generalize the relationship existing between various modules of the system.

In general, the level of details to be provided in the document depends on the type of system being developed and also on the process being implemented on this occasion.

By considering various aspects, IEEE has launched following format for generating the software requirements documents,

1. Introduction

- Purpose of requirements document
- Product scope
- Definitions, acronyms and abbreviations
- References
- Overview of rest of the requirements document

2. General description

- Product perspective
- Product functions
- User characteristics
- General constraints
- Assumptions and dependencies

3. Specific requirements

- Functional
- Non-functional
- External interface requirements
- System functionality and performance
- Logical database requirements
- Design constraints
- Emergent system properties
- Quality characteristics

4. Appendices

Here, the details of the application are included. The fundamental issues which can be addressed in this regard are hardware requirements as well as database requirements.

5. Index

Indexes are usually included to provide ease in locating appropriate entities. The most fundamental index is given in alphabetical order. There are many ways to represent indexes and any of them can be followed. The above mentioned format is a standard notation for developing the software requirement specification. But it is assumed that the above format defines only general characteristics when considered along with the organization developing the software. Hence, the

above format is further in matured and certain new specifications are added to it. This helps in directly applying preparation of SRS at the organization level. Hence, the improved format is given below,

1. Preface

Here, the fundamentals of the document are defined (say) the current version and its history. Also the details include the reasons for introducing the current documents along with the summary of changes made to the previous version in bringing up the current version.

2. Introduction

Summary of details under this side heading are given below,
The need for the system
The major functionality within the system and with other systems.
The way the given system remains compatible with the business objectives of the organization involved in developing the given system

3. Glossary

Glossary includes definitions for various technical terms used in requirements document.

4. User requirements definition

Here, usually the services delivered by a given system to the users are specified along with non-functional requirements of the system. This illustration is usually made in most elucidating style using the formal language constructs, diagrams or user understandable notations.

5. System architecture

Here, usually the overall system architecture is presented in an abstract format. This includes the distribution of modules throughout the architecture along with the relationships existing between them. The components (within the architecture) which are used more than once, should be represented along with the same signature (representation) so that they can be recognized easily.

6. SRS or System requirement specification

In this case both functional as well as nonfunctional requirements are described in a more detailed manner.

7. System models

One or more models depicting the relationship between the system, its environment and its components, along with their vicinity where they are highlighted. The most commonly used models in this case are semantic, data, dataflow and object models.

8. System evolution

In this case usually the assumptions based on which the current system is brought up is summarized. Also, the probable changes which can be made to the system on the account of hardware trends and future user needs are also addressed.

9. Appendices

Here, the details of the application are included. The fundamental issues which can be addressed in this regard are hardware requirements and database requirements.

10. Index

Indexes are usually included to provide ease in locating appropriate entities. The most fundamental index is given in alphabetical order. There are many ways to represent indexes and any of them can be followed.

Characteristics of a good software requirement specification

A good SRS document must possess the following characteristics

(i) Concise

The SRS document must be up to the point, unique and consistent because lengthy and irrelevant descriptions decrease the readability and increase the possibility of errors.

(ii) Organized

The SRS document must be well-organized because it becomes simple to understand and one can make the necessary changes easily as per the customer requirements.

(iii) Black-box view

The SRS document should represent only the physical behaviour of the system rather than describing its implementation issues. In other words the system which is being developed must be considered as a black box and its external behaviour should be defined. Due to this reason, the SRS document is also known as black-box specification of a system.

(iv) Conceptual integrity

A good SRS document must possess conceptual integrity so that the contents of the document can be understood easily by a reader.

(v) Response to undesired events

The document must describe the system responses to exceptional conditions.

(vi) Verifiable

All requirements of the system must be verified in order to determine whether all the requirements are met during implementation or not. If any feature is not verifiable then it should be mentioned separately in the "goals of implementation" section of the SRS document.

Role of software requirement specification in software engineering

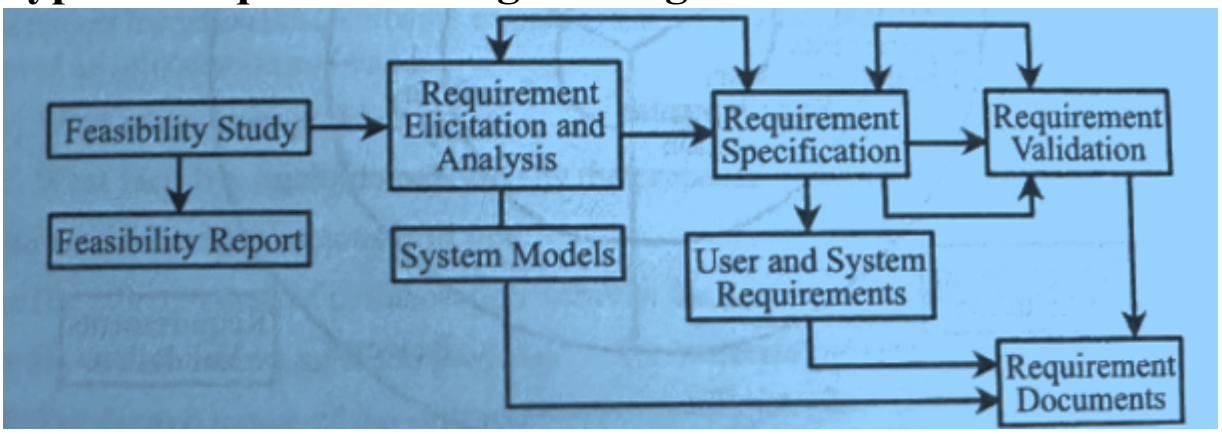
Software Requirement Specification (SRS) document is the final outcome that comes after problem analysis. But, both problem analysis and SRS are carried out simultaneously due to which these activities overlap with movement from both the activities to the other. The information regarding the SRS is obtained from analysis therefore, specification activity follows the analysis activity.

The formal modeling performed in problem analysis is not treated as SRS as it lays stress on the problem structure than the external behaviour. The things like modeling of user interfaces, minor issues, performance, design constraints, recovery etc., are rarely modeled. But these things should be specified in SRS, these all constraints are needed to be modeled to design a system easily and clearly. On the other hand, it is difficult for some structures to be translated into external behaviour specification due to their limit use.

Requirements Engineering

Requirements Engineering is the process of identifying, eliciting, analyzing, specifying, validating, and managing the needs and expectations of stakeholders for a software system.

Types of requirement engineering



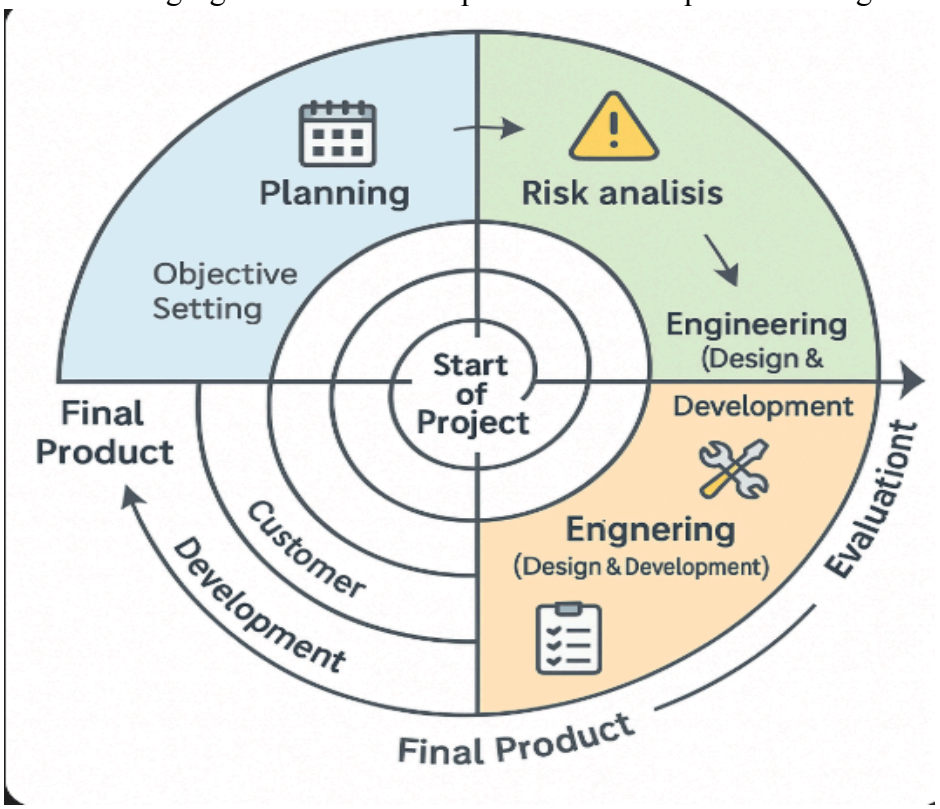
1. Feasibility Study
2. Requirements elicitation
3. Requirements specification

4. Requirements for verification and validation
5. Requirements management

Spiral model

This is an alternative method of the requirements engineering process. The spiral model is partitioned under three main activities i.e., requirements specifications, requirements validation and requirements elicitations. The time required in completion of given process depends on the type of system under development and also the process implemented.

The following figure describes the spiral model of requirements engineering process.



As shown in the above figure, the core of all processes will be the determination of business, functional and nonfunctional requirements specification. Hence, the developers usually pay higher attention at this stage. While certain other intermediate processes become closer to the final phase like review.

Feasibility Studies

The feasibility study mainly concentrates on below five mentioned areas below. Among these Economic Feasibility Study is the most important part of the feasibility analysis and the Legal Feasibility Study is less considered feasibility analysis.

Technical Feasibility

Technical feasibility assesses the current resources (such as hardware and software) and technology, which are required to accomplish user requirements in the software within the allocated time and budget. For this, the software development team ascertains whether the current resources and technology can be upgraded or added in the software to accomplish specified user requirement Technical feasibility performs the following tasks:
• It analyzes the technical skills and capabilities of the software development team member.
• It determines whether the relevant technology is stable and established or not.
• It ascertains that the technology chosen for software development has a large number of users so that they can be consulted when problems arise or improvements are required.

Operational Feasibility

Operational feasibility assesses the extent to which the required software performs a series of steps to solve business problems and user requirements. This feasibility is dependent on human resources (software development team) and involves visualizing whether or not the software will operate after it is developed and be operative once it is installed. It also performs the following tasks:
• It determines whether the problems anticipated in user requirements are of high priority or
• It determines whether the solution suggested by the software development team is acceptable or not.
• It analyzes whether users will adapt to a new software or not.
• It determines whether the organization is satisfied by the alternative solutions proposed by the software development team or not.

Economic Feasibility

Economic feasibility determines whether the required software is capable of generating financial gains for an organization or not. It involves the cost incurred on the software development team, estimated cost of hardware and software, cost of performing feasibility study, and so on. For this, it is essential to consider expenses made on purchases (such as hardware purchase) and activities required to carry out software development. In addition it is necessary to consider the benefits that can be achieved by developing the software.
A software is said to be economically feasible if it focuses on the issues listed below.
• Cost incurred on software development to produce long-term gains for an organization.
• Cost required to conduct full software investigation (such as requirements elicitation and requirements analysís).
• Cost of hardware, software, development team and training.

Legal Feasibility

Legal feasibility is an analysis performed to understand if the proposed plan conforms to the legal and ethical requirements. These requirements may involve zoning laws, data protection acts, or social medía laws, etc.
For example, your company is planning to open a branch in a new region. According to the studies you recognize that the country does not allow an individual foreigner owning a property Therefore you select the rental option instead of buying.

Scheduling Feasibility (Time Feasibility)

Completing a project on time is very important from an investor's perspective. Scheduling feasibility is a measure of how reasonable the project duration is. If the project takes longer than estimated, investors will have to bear extra costs.
For example, an investor proposed a hotel construction project to your company. However, he requested that the project will be completed in one year. The project team conducted a feasibility study based on a list of requirements to complete the project on time. After analyzing the legal, technical, economic, scheduling, and operational feasibility studies, you will identify any constraints the proposed project may face. Simply put, a project may face internal and external constraints.
Internal Constraints: Technical, technology and budget are an example of internal project constraints. Financial, marketing are examples of internal corporate constraints.
External Constraints: Laws, regulations, and environments are examples of external constraints.

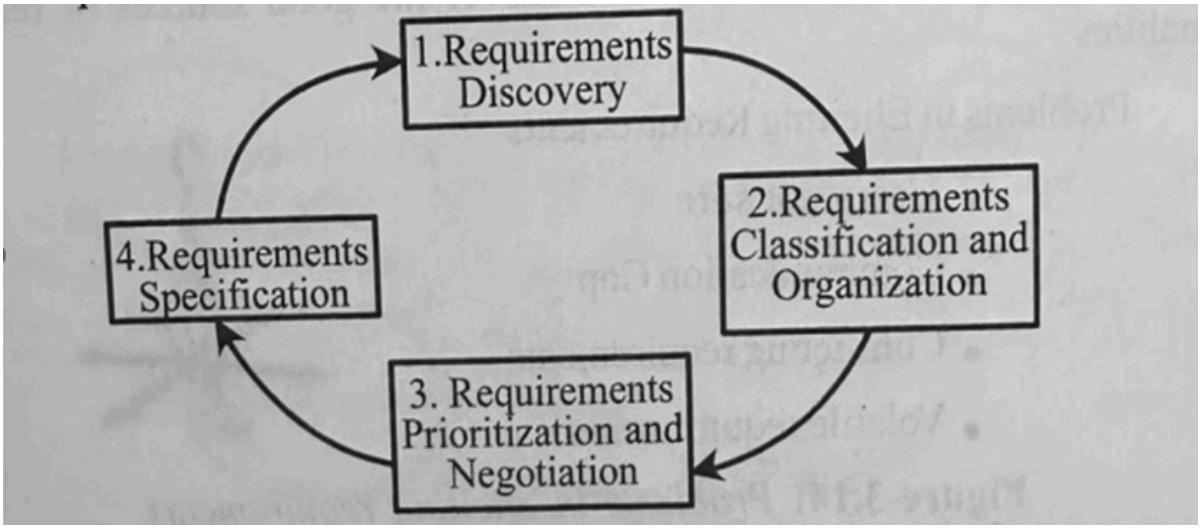
Advantages of Conducting a Feasibility Study

• Might uncover new ideas and opportunities.
• Provides inputs to improve decision making.
• Helps to prevent the project from risks. Enhances the success rate of the project.
• Lists the reasons to undertake the project.
• Lists the reasons not to undertake the project.
• Helps to define internal and external project constraints. Measures the ability and possibility to complete a project successfully.
• Emphasizes potential problems,
• Helps to develop marketing strategies.

Requirements elicitation

After an initial feasibility study, the next stage of the requirements engineering process is requirements elicitation and analysis. In this activity, software engineers work with customers and system end-users to find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints, and so on.

Requirements elicitation and analysis may involve a variety of different kinds of people in an organization. A system stakeholder is anyone who should have some direct or indirect influence on the system requirements. Stakeholders include end users who will interact with the system and anyone else in an organization who will be affected by it. Other system stakeholders might be engineers who are developing or maintaining other related systems, business managers, domain experts, and trade union representatives.



The activities involved in requirement elicitation and analysis include

- Requirements discovery
- Requirement classification and organization
- Requirement prioritization and negotiation
- Requirements specification

1. Requirements discovery

This is the process of interacting with stakeholders of the system to discover their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity. There are several complementary techniques that can be used for requirements discovery.

2. Requirements classification and organization

This activity takes the unstructured collection of requirements, groups related requirements, and organizes them into coherent clusters. The most common way of grouping requirements is to use a model of the system architecture to identify sub-systems and to associate requirements with each sub-system.

3. Requirements prioritization and negotiation

Inevitably, when multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation. Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.

4. Requirements specification

The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced.

The requirements elicitation and analysis is an iterative process with continual feedback from each activity to other activities. The process cycle starts with requirements discovery and ends with the

requirements documentation. The analyst's understanding of the requirements improves with each of the cycle. The cycle ends when the requirements document is complete.

Definition

Requirement elicitation and analysis is concerned with the process of analyzing requirements to:

- Detect and resolve conflicts between requirements.
- Discover the bounds of the software and how it must interact with its environment.
- Elaborate system requirements to derive software requirements.

Problems in elicitation and analysis

Unawareness

Lack of technical knowledge and unawareness of the technical aspects of the system from the stakeholder's side. Sometimes they may demand unrealistic things and they do not know what they exactly expecting from the system. Stakeholders express their requirements in the most general terms; it is difficult to find the technical aspects of the system from the general terms. Different people expect different services from the proposed system, requirements engineers has to discover the good sources of requirements and commonalities.

Problems in eliciting requirements

- Users not sure
- Communication gap
- Conflicting requirements
- Volatile requirements

Communication gap

Even if the end users and stakeholders are clear about the need for developing the new system, they may find it difficult to express it in a concrete manner due to their less exposure to the computing systems. They may state the requirements in an ambiguous or non-testable fashion. Sometimes the obvious basic requirements may be omitted and they concentrate on explaining the peripheral requirements. The system engineer needs to drive the stakeholders in the right direction throughout the requirements gathering stage so that they focus only on the most important requirements.

Conflicting requirements

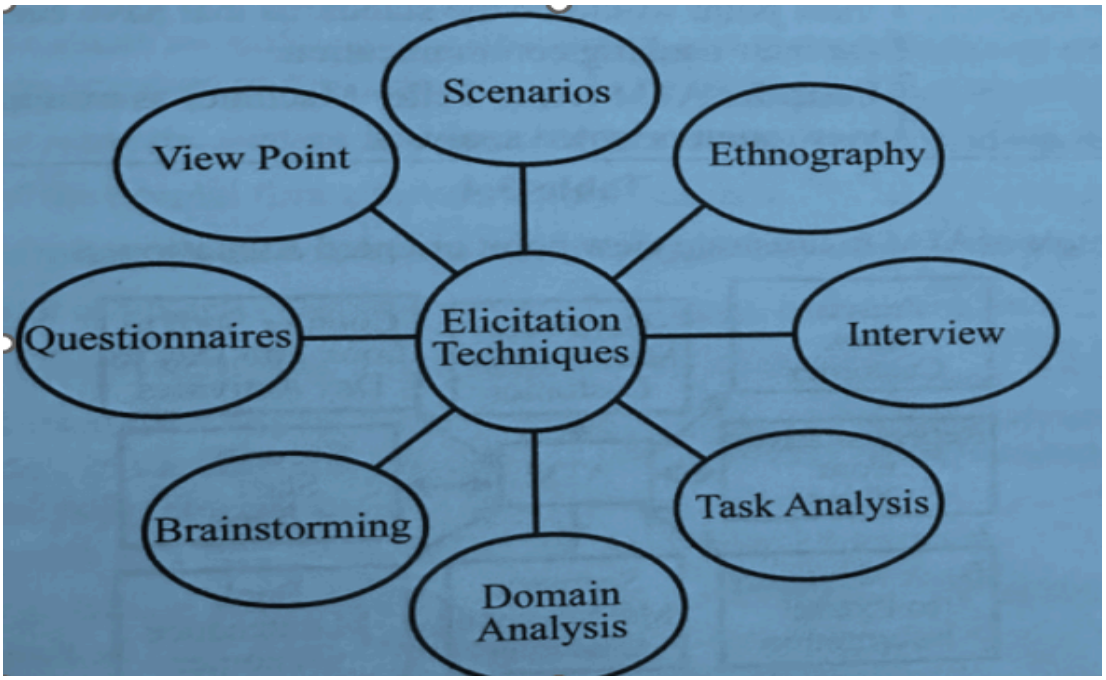
This problem increases with the number of stakeholders involved in the project. There may be conflicting needs and priorities for each stakeholder based on the business areas they are covering. For example, a stakeholder from the marketing team will try to provide requirements that will expedite the creation or new products, while the end users of the system will look for efficient screens that help easy data capture. During the analysis phase the requirements analyst and the client should collaboratively discuss and resolve any conflicting requirements from multiple stakeholders.

Volatile requirements

Requirements may get changed during the elicitation stage due to the entry of new stakeholders who have different perspectives of the system. Although this is helpful in clarifying the requirements better, it sometimes creates friction between the requirements engineer and the stakeholders due to continuous change in the scope.

Techniques for requirements elicitation and analysis

Once the requirements sources have been identified, the software engineer can start eliciting requirements from them. This topic concentrates on techniques for getting human stakeholders to articulate their requirements. It is a very difficult area and the software engineer needs to be sensitized to the fact that users may have difficulty in describing their needs, may leave important information unstated, or may be unwilling or unable to cooperate. It is particularly important to understand that elicitation is not a passive activity, and that, even if cooperative and articulate stakeholders are available, the software engineer has to work hard to elicit the right information.



Techniques for requirements elicitation and analysis are:

1. View point oriented elicitation
2. Scenarios
3. Interview
4. Ethnography
5. Task analysis
6. Domain analysis
7. Brainstorming

8. Questionnaires etc.

1. View point oriented

View point oriented elicitation approach takes different views given by different stakeholders. All complex systems should therefore be analyzed from a number of viewpoints. A key strength of view point oriented analysis is that it recognizes multiple perspective and provides a framework for discovering conflicts in the requirements proposed by different stakeholders. View points can be used as a way of classifying stakeholders and sources of requirements.

There are three generic types of view points:

- **Interactor view points**: Represents people or other systems that interact directly with the system.
  Example: In a bank ATM system examples of interactor view points are bank's customers and bank's account database.

- **Indirect view points**: Represents stakeholders who do not use the system themselves but who influence the requirements in some way.
  Example: In the bank ATM system examples of indirect view point are management of the bank and the bank security staff.

- **Domain view points**: Represent domain characteristics and constraints that influence the system requirements.
  Example: In the bank ATM system an example of a domain view point would be the standards that have been developed for inter banking communication.

2. Scenario

The scenario technique is a requirement elicitation method that involves describing real-world situations in which users interact with the system. It helps stakeholders visualize how the system should behave in different conditions, leading to better understanding and identification of requirements.

3. Interviews

The objective of conducting an interview is to understand the customer's expectations of the software.

It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility. Interviews may be open-ended or structured.

In open-ended interviews, there is no pre-set agenda. Context-free questions may be asked to understand the problem.

In a structured interview, an agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

4. Ethnography

Software systems do not exist in isolation—they are used in a social and organizational context, and software system requirements may be derived or constrained by that context. Satisfying these social and organizational requirements is often critical for the success of the system. One reason why many

software systems are delivered but never used is that they do not take proper account of the importance of these requirements.

Definition

"Ethnography is an observational technique that can be used to understand social and organizational requirements."

In ethnography, a group of people are studied in their natural settings. The users' requirements can be well understood by participating in their day-to-day activities for a certain period of time.

The difference between the assumed work and the actual work is most important. Thus, the ethnography is particularly effective at discovering two types of requirements:

1. Requirements that are derived from the way in which people actually work rather than the way in which definitions say ought the work.
2. Requirements that are derived from cooperation and awareness of other people's activities.

Ethnography may be combined with prototyping. The following diagram shows the process of requirement analysis which combines ethnography and prototyping.

5. Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

6. Domain analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

7. Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

8. Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

Requirements specification

Requirements collected get transformed into structured SRS document. It defines the requirements of the proposed system. The requirement consists of end user's crude requirements and detailed description of the system requirement. It focuses on the functionality of the system.

Specification is very important as it clearly indicates what the requirements, to develop the project are.

Requirements specification adds on further information to the requirements definition.

Specification is presented with the system models, useful for designing the software. It includes all the specification and constraints on the operations of the system.

Requirement specification methods

There are five different specification methods:

- **Structured natural language**: Defining standard form or template to express the requirement specification using natural language.
- **Design description language**: This method concentrates on operations and constraints of the system. It makes use of programming languages with more abstract features.
- **Requirement specification language**: Various special purpose languages have been designed to express software requirement such as PSL/PSA, Problem Statement Language (PSL), Problem Statement Analyzer (PSA). The advantages of this is special purpose tool support can be developed.
- **Graphical notations**: Best known graphical notation for requirement is SADT. Structured Analysis and Design Technique. It has many complex graphical vocabulary and is mostly used by specialists. Graphical notations help the analyst to read and understand the requirement quickly.
- **Mathematical specifications**: These notations based on formal mathematical concept such as finite state machine or more basic concept such as sets.

Requirements validation techniques

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.

Requirement validation techniques

With other techniques to check the entire system or parts of the system. The selection of the validation technique depends on the appropriateness and the size of the system to be developed.

Some of these techniques are:

- **Test Case Generation**: The requirements specified in the SRS document should be testable. The test in the validation process can reveal problems in the requirement. In some cases test becomes difficult to design, which implies that the requirement is difficult to implement and requires improvement.
- **Automated Consistency Analysis**: If the requirements are expressed in the form of structured or formal notations, then CASE tools can be used to check the consistency of the system. A requirements database is created using a CASE tool that checks the entire requirements in the database using rules of method or notation. The report of all inconsistencies is identified and managed.

- **Prototyping**: Prototyping is normally used for validating and eliciting new requirements of the system. This helps to interpret assumptions and provide an appropriate feedback about the requirements to the user. For example, if users have approved a prototype, which consists of graphical user interface, then the user interface can be considered validated.

Note: The requirement reviews can be either formal or informal.

Types of checks during requirements validation

During the requirements validation process, different types of checks should be carried out on the requirements in the requirement document. These checks include:

Validity checks

A user may think that a system is needed to perform functions. Systems across the user community.

Consistency checks

Requirements in the document should not conflict (i.e.) there should not be contradictory constraints or different description of the same system function.

Completeness checks

The requirements document should include requirements which define all functions and constraints intended by the system user.

Realism checks

Using knowledge of existing technology, the requirements should be checked to ensure that they can actually be implemented. It also takes into account of the budget and schedule for the system development.

Verifiability

The system requirements should always be written so that they are verifiable.

Requirement management

Introduction

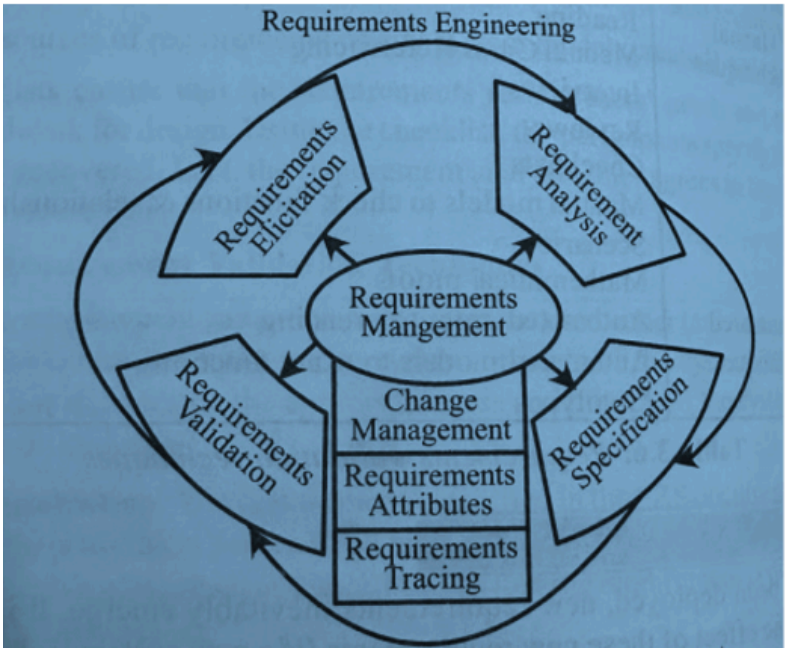
Once a system has been deployed, new requirements inevitably emerge. It is difficult for the users to anticipate the effect of these new requirements if a new system is developed for these requirements on the organization. Thus, to understand and control changes to system requirements, requirements management is performed.

Enduring and volatile requirements

Requirements classes

1. **Enduring requirements**: Stable requirements which are derived from the core activity of the organization and which relate directly to the domain of the system.
2. **Volatile requirements**: These requirements are likely to change during the system development or after the system has been put into operation.

- **Mutable requirement**: Requirements which change because of changes to the environment in which the organization is operating.
- **Emergent requirement**: Requirements which emerge as the customer's understanding of the system develops during the system development.
- **Consequential requirement**: Requirements which result from the introduction of the computer system which may change the organization's process and open up for new system requirements.
- **Compatibility requirements**: Requirements which depend on the particular systems.



Requirements management can be defined as a process of eliciting, documenting, organizing business processes within an organization, and controlling changes to the requirements.

Activities in requirements management

Generally, the process of requirements management begins as soon as the requirements document is available, but planning for managing the changing activities performed in requirements management are:

- Establishing a relationship amongst stakeholders and involving them in the requirements.
- Recognizing the need for change in the requirements.
- Identifying and tracking requirements attributes.

Requirements management enables the development team to identify, control, and track requirements and changes that occur as the software development process progresses.

Advantages

- **Better control of complex projects**: This provides the development team with a clear understanding of what, when and why the software is to be delivered. The resources are allocated according to user-driven priorities and relative implementation effort.
- **Improves software quality**: This ensures that the software performs according to enhance software quality. This can be achieved when the developers and testers have a precise understanding of what to develop and test.
- **Reduced project costs and delays**: This minimizes errors early in the development. As a result, the project costs also reduce.
- **Improved team communications**: This facilitates early involvement of users to ensure that their needs are achieved.
- **Easing compliance with standards and regulations**: This ensures that standards involved with software compliance and process improvement have a thorough understanding of requirement management. For example, CMM addresses requirements management as all the user requirements are specified in the software requirement specification. The project manager, as part of requirements management, tracks the requirement for the current project and those which are planned for the next release.

Requirements management process

Requirements management starts with planning, which establishes the level of requirements management needed. After planning, each requirement is assigned a unique identifier so that it can be crosschecked by other requirements. Once requirements are identified, requirements tracing is performed.

Requirement tracing

Requirement tracing is a medium to trace requirements from the start of development process till the software is delivered to the user. The objective of requirement tracing is to ensure that all the requirements are well understood and included in test plans and test cases.

Various advantages of requirement tracing are:

- It verifies whether user requirements are implemented and adequately tested or not.
- It enables user understanding of impact of changing requirements.

Traceability techniques facilitate the impact of analysis on changes of the project, which is under development. Traceability information is stored in a traceability matrix, which relates requirements to stakeholders or design module. The traceability matrix refers to a table that correlates high-level requirements with the detailed requirements of the product.

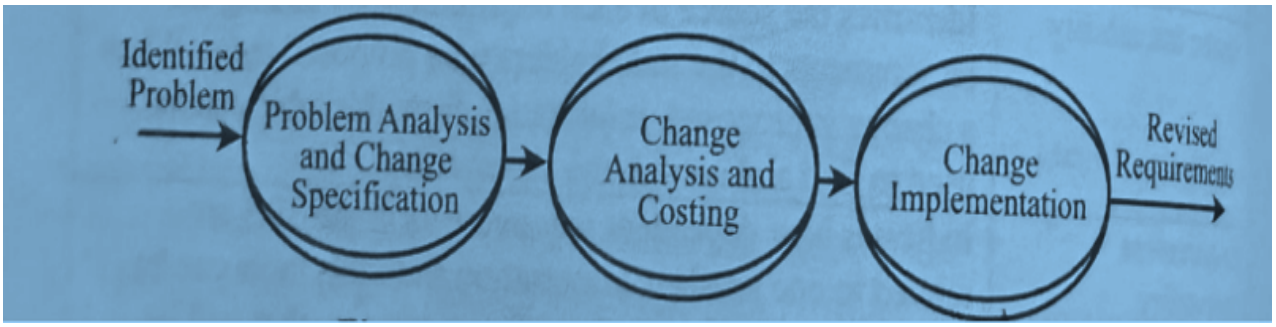
Traceability

“Traceability” refers to the ability to track and link a requirement throughout its entire lifecycle, from its initial definition to its implementation and testing.

- **Features traceability**: Indicates how requirements relate to important features specified by the user.
- **Source traceability**: Identifies the source of each requirement by linking the requirements to the stakeholders who proposed them. When a change is proposed, information from this table can be used to find and consult the stakeholders.
- **Requirement traceability**: Indicates how dependent requirements in the SRS are related to one another. Information from this table can be used to evaluate the number of requirements that will be affected due to the proposed changes(s).
- **Design traceability**: Links the requirements to the design modules where these requirements are implemented. Information from this table can be used to evaluate the impact of proposed requirements changes on the software design and implementation.
- **Interface traceability**: Indicates how requirements are related to internal interface and external interface of a system.

Requirements change management

Requirements change management is used when there is a request or proposal for a change in the requirements. The advantage of this process is that the changes to the proposals are managed consistently and in a controlled manner.



Stages of requirements change management

An efficient requirements change management process undergoes a number of stages for changes to the requirement. These stages are:

- **Problem analysis and change specification**: The entire process begins with identification of problems to the requirements. The problem or proposal is analyzed to verify whether the change is valid or not. The outcome of the analysis is provided to the ‘change requester’ and a more specific requirements change proposal is then made.
- **Change analysis and costing**: The effect of a change requested on the requirement is assessed according to traceability information. The cost for this can be estimated on the basis of modification made to the design and implementation. After the analysis is over, a decision is made whether changes are to be made or not.
- **Change implementation**: Finally, the changes are made to the requirements document, system design, and implementation. The requirements document is organized in such a manner so that changes to it can be made without extensive rewriting. Minimizing the external references and making document sections modular achieves changeability in the document.