CHAPTER # 03: REQUIREMENTS ANALYSIS

Requirements Analysis: An Overview

The purpose of requirements analysis is to establish an understanding of the application domain, capture, formalize, analyze, and validate user requirements on the system to be built. Once requirements have been gathered (or elicited), the requirements analysis phase is initiated. This phase aims to investigate the gathered requirements in detail, categorize them, organize them into related subsets, examine them for consistency, omissions, and ambiguity, and rank them based on the needs of the customer/users. Requirements analysis acts as a bridge between system engineering and software design.

To build something, it's crucial to first understand what that "something" is meant to be. This process of understanding and documenting these requirements is called requirements analysis. Requirements generally express what an application is meant to do, without specifying how to accomplish these functions. The key objective of requirements analysis is to describe the requirements in terms of relationships, provide a basis for design, and serve as a basis for validation for the software after it is built. The requirements analysis phase starts in parallel with requirements elicitation and involves refining and modeling the requirements to identify inconsistencies, errors, omissions, and other defects. Requirements analysis is usually done with the use of one or more system models that present an abstract description of the system. These models also act as a bridge between users, customers, and developers, as they are easily understandable by all parties. The output of requirements analysis is a document generally referred to as software requirement specification (SRS).

Levels Or Categories Of Requirements Analysis

Requirements analysis is divided into two levels:

1. Customer or C-requirements
2. Developer (Detailed) or D-requirements

The first level documents the customer's wants and needs, expressed in language clear to them, and is referred to as customer requirements or C-requirements. The second level documents the requirements in a specific structured form, known as developer requirements or D-requirements.

Customer Or C – Requirements

When the development community begins requirements analysis, the customer is typically still forming concepts of what they want and need. The analysis of requirements is a person-to-person activity, carefully organized to produce the best application. Software engineers gather the customer or C-requirements through various techniques, with interviews being the primary method.

Road Map Of C- Requirements

The following figure shows the different steps of gathering the C-requirements. The first step is to identify the "Customer or User", then conduct the interview with the customer which is the main technique used to gather the C-requirements which identifies what the customer actually wants. After identifying the needs of the customer, the analyst is now able to write the C-requirements in a standard document. On that basis, the analyst will be further able to build the D-requirements that are the main purpose of writing the C-requirements.

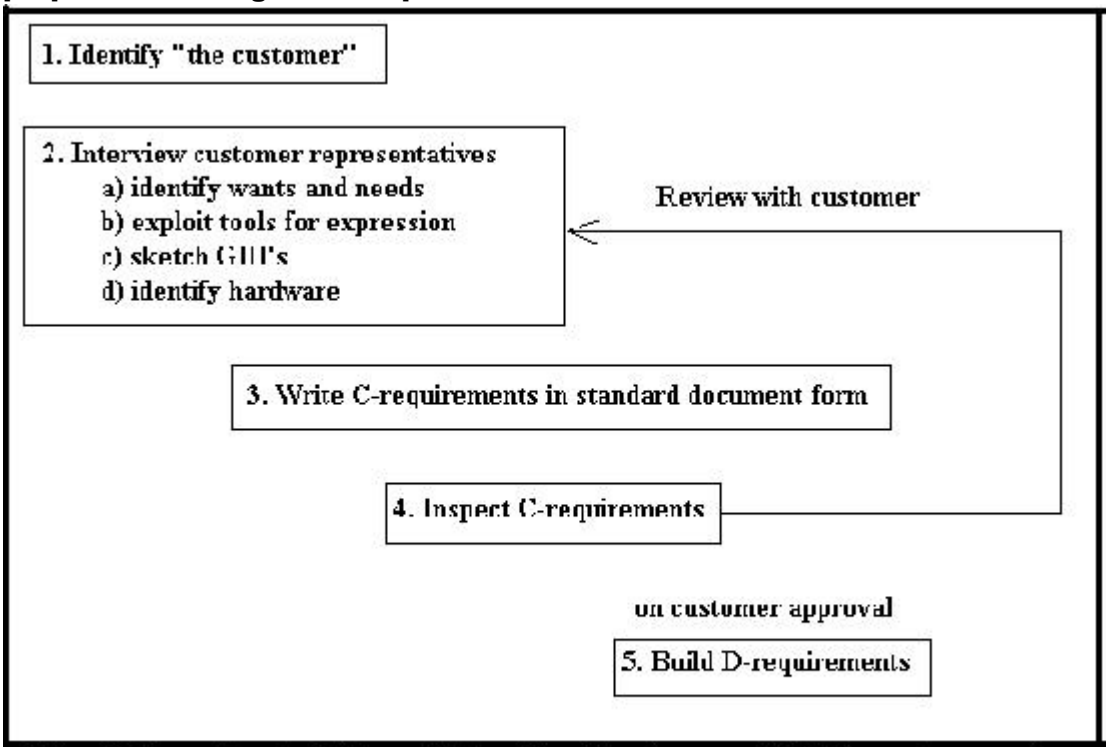


Fig. 3.1 Typical Road Map for Customer (C-) Requirements

There are various ways to express the C-requirements:
- If the requirement is simple and stands alone, express it in clear sentences within an appropriate section of the SRS.
- If the requirement is an interaction between the user and the application, express it via a use case.
- If the requirement involves process elements, each taking inputs and producing outputs, then use the data flow.

- If the requirement involves states that are going to change at some particular event, then express these types of requirements via STD (state transition diagram).

Detailed or D – Requirements

Software engineers need a base for design and implementation; this base consists of the detailed requirements, also called "specific requirements," "functional requirements," "developer requirements," or "D-requirements". D-requirements consist of a complete list of specific properties and functionality that the application must possess, expressed in final detail. Each of these requirements is numbered, labeled, and tracked through implementation. They are consistent with and elaborate upon the C-requirements. The D-requirements are intended to be read primarily by developers. Customers are interested in them as well and are typically able to understand and comment on many of them.

Road Map Of D- Requirements

The following figure shows a typical sequence of activities for gathering and documenting D-requirements. There are five steps to obtain the D-requirements. The first step describes the ways in which specific requirements can be organized. Then create the sequence diagrams, after that, the third step is started in which the D-requirements are written from the C-requirements. Then we begin writing tests for each of the specific requirements simultaneously with writing the requirements themselves. Although D-requirements are written primarily for developers and their tests are also reviewed with the customer. Then finally the D-requirements should then be inspected and released.

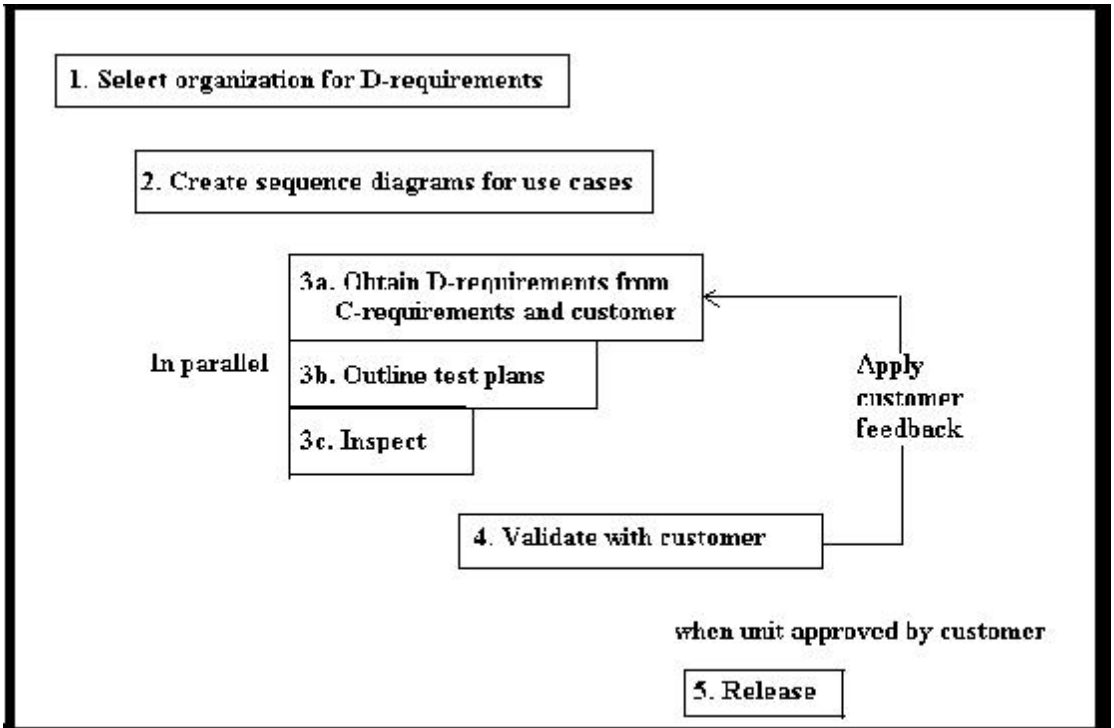


Fig. 3.2 Typical Road Map for Detailed (D-) Requirements

Once the D-requirements have been collected, the project documents are updated or reflect the improved project knowledge.

- Organize the specific Requirements.
- Write D Req from C Req.
- Test and inspect for each specific Req.
- Release final Req.

The D-Requirements ("developer" or "Detailed" requirements) are written primarily for designers and developers. They are created from C-requirements, as well as from continued customer interaction. The D-requirements must be testable, traceable, and consistent.

Types Of D – Requirements

There are several types of requirements. This classification applies to both C- and D- requirements. During the writing of C-requirements, these distinctions are often secondary to getting points across to the customer about the application in general. The classification becomes much more significant when writing the D-requirements, however, because it guides the development and testing process in different ways.

1. Functional Requirements
2. Non-Functional Requirements
3. Design Constraints

Functional Requirements

Functional requirements specify services that the application must provide. They are associated with specific functions, tasks, or behaviors of the system that must support these requirements. Functional requirements address the quality characteristic of functionality according to the ISO quality standards.

Functional requirements collectively define what a system does, typically in terms of visible changes that can be effected in the system or that it can cause in the outside world. They are usually action-oriented ("When the user does x, the system will do y."). Most products and applications, conceived to do useful work, are rich with software functional requirements. Software is used to implement the majority of the functionality.

Non-Functional Requirements

Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless. They define system properties and constraints such as reliability, response time, and storage requirements. Non-functional requirements specify timing constraints that the application must observe and include aspects like performance, reliability, security, usability, error handling, efficiency, information, interface, service, economy, maintainability, and constraints.

Design Constraints

Design constraints typically impose limitations on the design of the system or the processes used to build a system. They describe limits or conditions on how the application is to be designed or implemented. These requirements are not intended to replace the design process but specify conditions imposed upon the project by the customer, the environment, or other circumstances. Design constraints can be found in the developmental infrastructure surrounding the system to be developed, including operating environments, compatibility with existing systems, application standards, and corporate best practices.

Methodologies And Tools Used For Requirements Analysis

Various methodologies and tools are used for requirements analysis, including data-flow diagrams, entity-relationship diagrams, object-class models, state-transition diagrams, and use cases.

DFD (Data – Flow Diagrams): DFDs are used to study the manner in which information enters a system and how it is transformed as it flows through the system. They graphically represent the system using symbols for data source/destination, process, data flow, and data store.

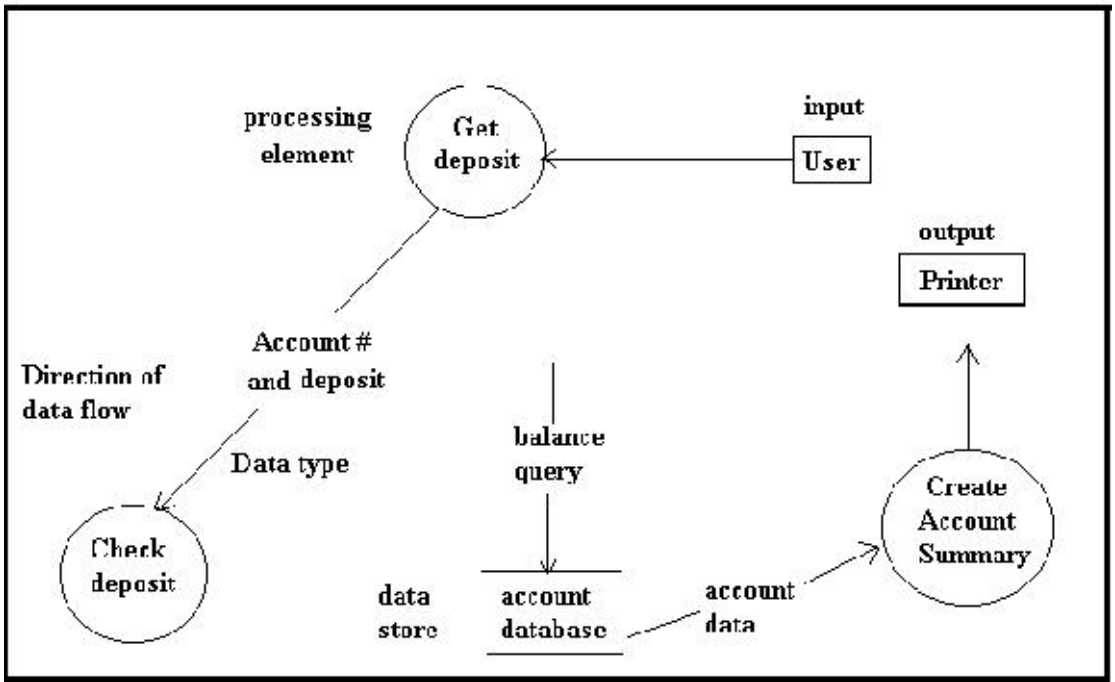
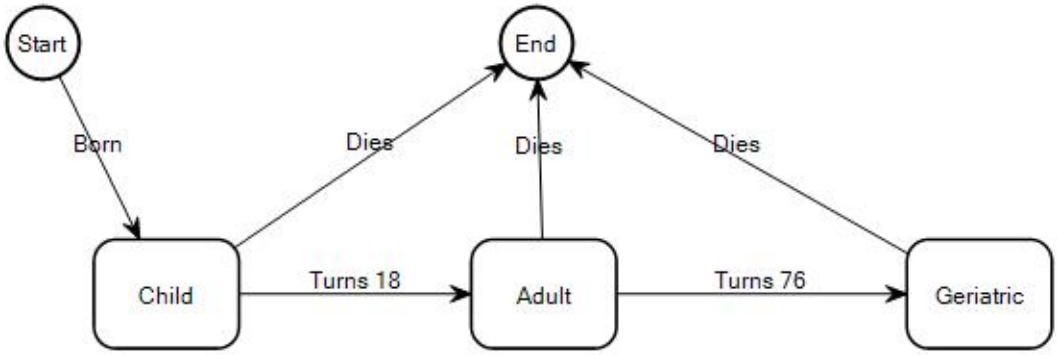
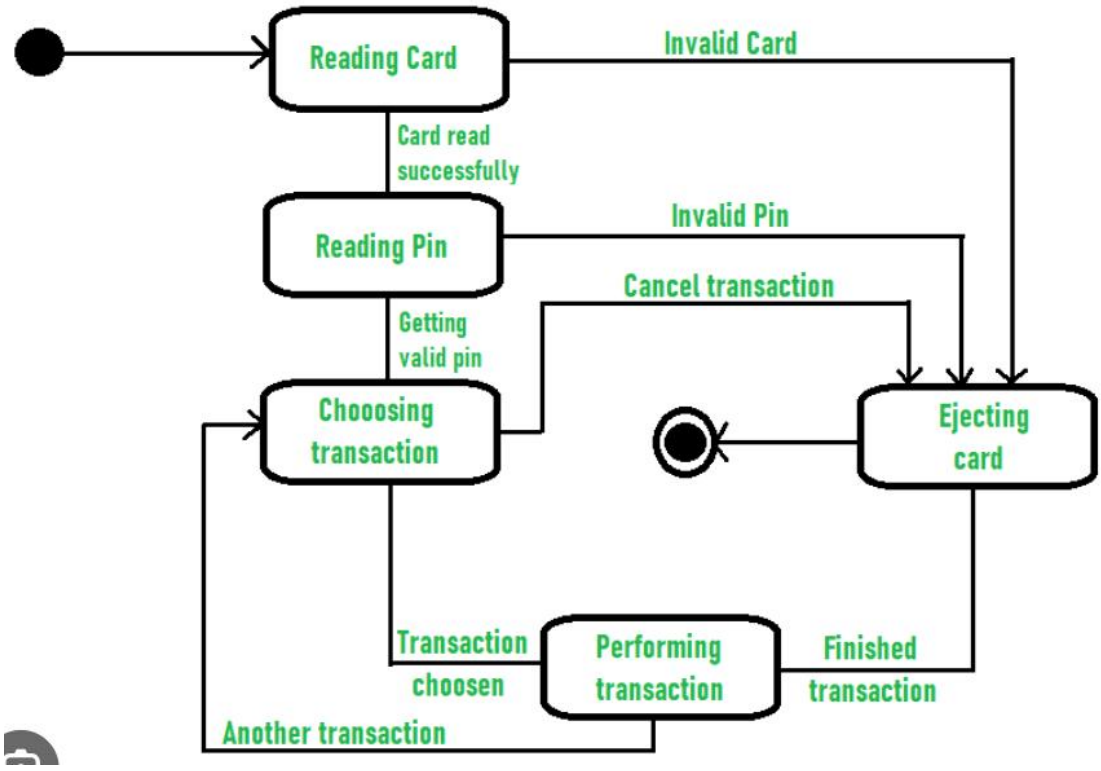


Fig. 3.3 Data Flow Diagram: Explanation of Symbols

STD (State Transition Diagrams): STDs are behavioral models of the system that indicate how the proposed system will behave in response to external events. They represent various "states" of the system and the manner in which transitions will be made from one state to another on receiving a stimulus.





Use-Cases: Use cases describe the interactions between the system and an external actor, representing a person, group, or another system. They provide a high-level visual representation of user requirements and describe all the steps that the actor and the proposed system need to perform to achieve a desired objective.

In conclusion, requirements analysis is a critical phase in software development, involving understanding, capturing, formalizing, analyzing, and validating user requirements. It ensures that the developed software meets the needs and expectations of its users.

Types of NFR

Non-functional requirements encompass various aspects that are critical for the overall success and usability of a software system. Let's delve deeper into each category:

1. Performance Requirements:

Performance requirements specify the system's expected performance metrics, such as throughput rate and response time. These requirements are particularly crucial for real-time applications where actions must complete within specified time limits. For example:

- **What is the acceptable throughput rate?**
- **What is the acceptable response time for different operations?**

2. Reliability and Availability Requirements:

Reliability requirements quantify the system's expected reliability, while availability requirements specify the degree to which the system should be available to its users. For instance:

- How often should the system experience failures?

- What is the acceptable downtime for maintenance?

3. Usability:

Usability requirements focus on making the system easy to use and understand for its intended users. They ensure that the interface is clean, intuitive, and user-friendly. Questions to consider include:

- How easy is it for users to navigate the system?

- Are there clear instructions and prompts provided to users?

4. Control or Security Requirements:

Control requirements define the level of access control and security measures that must be implemented in the system to protect sensitive information and ensure data privacy. Examples include:

- Must access to the system or information be controlled through authentication?

- What are the privacy requirements for user data?

5. Error Handling:

Error handling requirements specify how the system should respond to errors, both internally generated errors and errors from external sources. For instance:

- How should the system handle unexpected input or invalid data?

- What actions should be taken if a critical error occurs?

6. Efficiency Requirements:

Efficiency requirements focus on optimizing resource usage and minimizing waste within the system. They address concerns such as redundant processes and resource utilization. Questions to consider include:

- Are there any duplicate steps in the system that need to be eliminated?

- How can resource usage be optimized to improve efficiency?

7. Information Requirements:

Information requirements specify the data that is pertinent to users in terms of content, timeliness, accuracy, and format. They address questions such as:

- What inputs and outputs are necessary for the system?

- What are the required data storage and retrieval mechanisms?

8. Interface Requirements:

Interface requirements describe how the system communicates with its environment, including communication protocols, hardware interfaces, software compatibility, and user interface design. Examples include:

- What communication protocols and networks will the system use?
- How should the user interface be designed to enhance usability?

9. Service Requirements:

Service requirements define the needs for the system to be reliable, flexible, and expandable, including considerations for user demographics, training, distribution, and documentation. Questions include:

- Who will use the system, and what are their needs?
- How should the system be packaged, distributed, and documented?

10. Economy Requirements:

Economy requirements focus on cost-effectiveness and budget constraints associated with the system's development, deployment, and maintenance. Examples include:

- What are the budgetary limits for the project?
- How can costs be reduced or profits increased through system improvements?

11. Constraints:

Constraints impose limitations on the design or implementation of the system, such as technical, business, or contractual obligations. Examples include:

- Are there any specific hardware or software constraints that must be considered?
- What standards or regulations must the system adhere to?

Each of these non-functional requirements plays a crucial role in shaping the overall quality, usability, and performance of the software system, ensuring that it meets the needs and expectations of its users and stakeholders.

Chapter 4: Requirements Documentation

Requirements Documentation: An Overview

Documentation, such as hardcopy manuals or online help files, portrays the system's use and operation. Requirements documentation, typically in the form of a Software Requirements Specification (SRS), is crucial as it serves as a lasting impression of the requirements elicitation and analysis process. It forms the basis for development, testing, and understanding the system for various stakeholders. The SRS includes all requirements and should not assume any requirement as obvious.

The objective of documentation is to allow an independent person to review the system and understand the basis for its coding. A well-written

SRS defines planned features and functions, describes qualities like usability, and identifies existing solutions and competitive products.

Characteristics of SRS Document

A good SRS document should possess several characteristics:

1. Completeness: All functional and non-functional requirements should be included. Missing requirements are difficult to track and may lead to incomplete systems.

2. Clarity: The document should be unambiguous and understandable to all stakeholders. Ambiguities in natural language should be minimized, and the document may use requirement specification languages or modeling techniques for clarity.

3. Correctness: Every requirement stated in the SRS should be necessary for the proposed system. Stakeholder reviews and tracing requirements from preceding documents can help ensure correctness.

4. Consistency: Requirements at all levels must be consistent with each other. Conflicts between requirements must be identified and resolved.

5. Modifiable: The SRS should facilitate easy and consistent modification. Labels and minimal redundancy aid in highlighting and tracing changes.

6. Trace ability: Requirements should be labeled or numbered for trace ability through design, testing, and implementation phases. Backward and forward trace ability are crucial for understanding the system's evolution.

7. Feasibility: In feasible requirements should be identified and eliminated from the SRS, with reasoning provided.

8. Test ability: Requirements should be stated in a way that enables testers to create test plans and verify their fulfillment. Non-testable statements should be identified and revised.

Contents of SRS Document

The SRS document typically includes the following sections:

1. Introduction: Sets the tone, describes the purpose, scope, overview, and other relevant details of the document.

2. System Description: Describes the current system, proposed system goals, user characteristics, and assumptions.

3. Product Functions: Describes the general functionality of the product, including modules and detailed requirements.

4. Similar System Information: Describes relationships with other products, standalone or component nature, and any dependencies.

5. User Characteristics: Describes user features and expertise with software systems and the application domain.

6. User Objectives: Lists user objectives and requirements, including a "wish list" of desirable characteristics.

7. General Constraints: Lists general constraints affecting the design, such as speed, industry protocols, or hardware limitations.

8. Functional Requirements: Details the system's functionality, often partitioned into modules with associated requirements.

9. Non-Functional Requirements: Specifies system properties, constraints, and quality attributes.

10. Interface Requirements: Describes how the software interfaces with other software, users, hardware, and networks.

11. Design Constraints: Lists limitations or constraints likely to affect the design process.

12. Project Requirements: Includes project plans, schedules, budgets, and acceptance criteria.

13. Preliminary Schedule: Provides an initial project plan, including major tasks, dependencies, and resource requirements.

14. Preliminary Budget: Provides an initial budget breakdown by cost factor.

15. Appendices: Provides additional useful information, such as definitions, acronyms, references, etc.

Importance of Requirements Documentation

Documentation plays a crucial role in software development, aiding in understanding, development, and maintenance of the system. A well-documented SRS improves program quality, usability, and maintainability, ultimately reducing long-term software-related expenses. It allows stakeholders to review the system, understand user requirements, and facilitate system development and testing. Documentation should not be an afterthought but an integral part of the software development process.

In summary, requirements documentation, particularly in the form of a comprehensive SRS, is essential for successful software development. It should exhibit characteristics like completeness, clarity, correctness,

consistency, modifiability, traceability, feasibility, and testability to ensure its effectiveness.

Presentation Topic : 01

Risk Analysis in Software Requirements Engineering

Risk analysis in software requirements engineering is a critical process aimed at identifying, assessing, and mitigating potential risks that may impact the success of a software project. It involves several key steps and techniques to ensure that risks are properly managed throughout the software development life cycle.

Process involves :

1. Risk Identification:

- The first step in risk analysis is to identify potential risks that may arise during the development process. This can be done through various techniques such as brainstorming, documentation review, interviews with stakeholders, and historical data analysis.

2. Risk Assessment:

- Once potential risks are identified, they are assessed based on their potential impact on the project objectives and their likelihood of occurrence. This assessment helps prioritize risks and allocate resources accordingly.

3. Risk Prioritization:

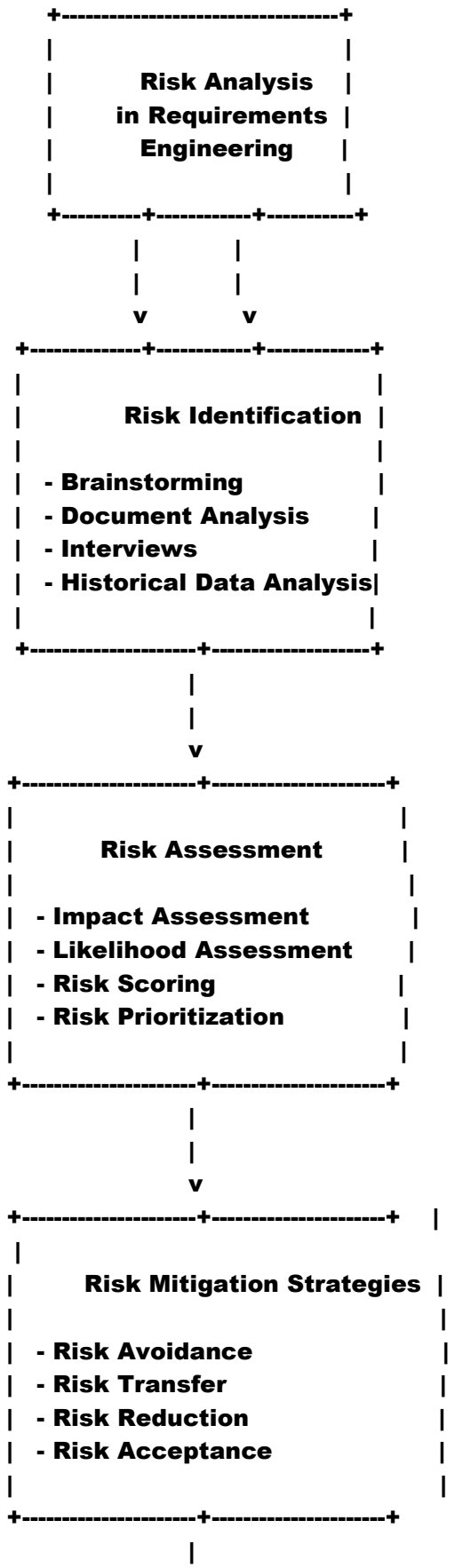
- Risks are prioritized based on their severity and likelihood, typically using techniques like risk matrices or risk scoring models. This helps focus efforts on addressing the most critical risks first.

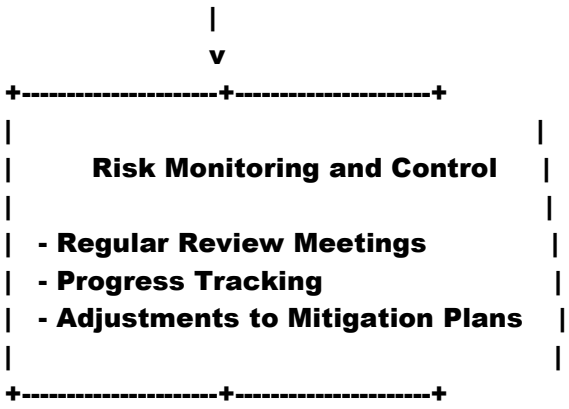
4. Risk Mitigation Strategies:

- After prioritizing risks, mitigation strategies are developed to address them. These strategies may include risk avoidance, risk transfer, risk reduction, or risk acceptance. The goal is to minimize the impact of potential risks on the project.

5. Risk Monitoring and Control:

- Throughout the software development life cycle, risks are continuously monitored and controlled to ensure that mitigation strategies are effective and to identify any new risks that may arise. This involves regular review meetings, progress tracking, and adjustments to mitigation plans as needed.


Risk Analysis in Requirements Engineering
Risk Identification
- Brainstorming
- Document Analysis
- Interviews
- Historical Data Analysis
Risk Assessment
- Impact Assessment
- Likelihood Assessment
- Risk Scoring
- Risk Prioritization
Risk Mitigation Strategies
- Risk Avoidance
- Risk Transfer
- Risk Reduction
- Risk Acceptance



Presentation topic : 02

Title: An Overview of Requirements Trace ability

Introduction:

Requirements trace ability is a fundamental aspect of requirements management, crucial for ensuring the successful development and deployment of software systems. It involves tracking the life of a requirement from its inception through to its implementation, refinement, and iteration. This overview delves into the essence of requirements traceability, its significance, types, techniques, and benefits.

Definition:

Requirements traceability, often abbreviated as RT, refers to the ability to describe and follow requirements in both forward and backward directions.

Significance:

Requirements traceability empowers software engineers to:

- Establish a clear list of system requirements.
- Develop user manuals through use case scenarios.
- Ensure every aspect of design corresponds to at least one user requirement.
- Trace each requirement to its fulfillment within the design.
- Maintain alignment between design and requirements throughout critical reviews.

Role in Verification:

Requirements traceability is not only a management tool but also supports verification activities by:

- Providing insights into the impact of requirement changes.
- Locating where requirements are implemented.
- Confirming allocation of all requirements.
- Identifying the needs addressed by each requirement.
- Validating the necessity of each requirement.
- Tracing design decisions influencing implementation.
- Ensuring compliance of implementation with requirements.

Types of Traceability:

1. Backward-from Traceability: Links requirements to their sources in other documents or individuals.
2. Forward-from Traceability: Links requirements to design and implementation components.
3. Backward-to Traceability: Links design and implementation components back to requirements.
4. Forward-to Traceability: Links other relevant documents to requirements.

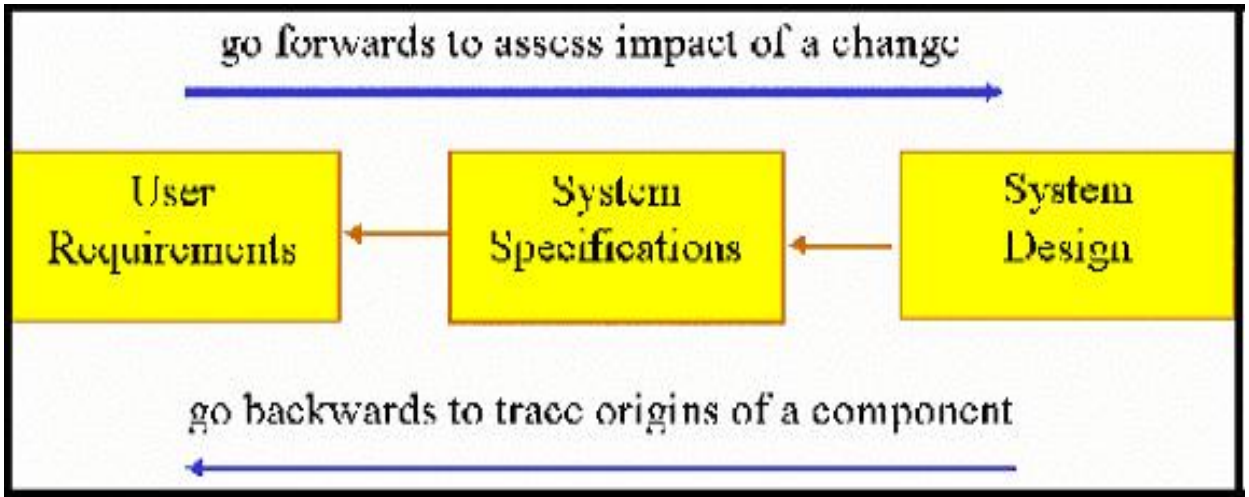


Fig. 7.1 Backwards / Forwards Tracability

Types of Requirements Traceability:

1. Requirements-Sources Traceability: Links requirements to the people or documents specifying them.
2. Requirements-Rationale Traceability: Links requirements to descriptions explaining why they are specified.
3. Requirements-Requirements Traceability: Links requirements to other dependent requirements.
4. Requirements-Architecture Traceability: Links requirements to implementing subsystems.

5. Requirements-Design Traceability: **Links requirements to specific hardware or software components.**
6. Requirements-Interface Traceability: **Links requirements to external system interfaces.**

Techniques for Traceability:

1. Traceability Tables: **Show relationships between requirements or between requirements and design components.**

Depends-on

	R1	R2	R3	R4	R5	R6
R1			×	*		
R2					*	×
R3				*	*	
R4		*				
R5						×
R6						

Fig. 7.2 Traceability Tables

2. Traceability Lists: **Simple lists showing relationships.**

Requirement	Depends-on
R1	R3, R4
R2	R5, R6
R3	R4, R5
R4	R2
R5	R6

Fig. 7.3 Traceability List

3. Automated Traceability Links: **Utilize databases to maintain requirements and traceability links.**

Traceability Policies:

- Define what traceability information should be maintained.
- Specify techniques and processes for maintaining traceability.
- Describe when traceability information should be collected.
- Assign roles and responsibilities for managing traceability.
- Provide guidelines for handling and documenting policy exceptions.

Benefits of Requirements Traceability:

- Ensures all requirements are designed and tested.
- Prevents unnecessary design features.
- Facilitates effective impact analysis for change management.

Conclusion:

Requirements traceability is indispensable for ensuring software systems meet their intended objectives. By establishing clear links between requirements, designs, and implementations, traceability enhances the quality, reliability, and maintainability of software systems throughout their lifecycle.

Note : For Presentation topic 03 see the ppt of Requirement Change Management and for Mids related content see the pdf of Mids preparation.