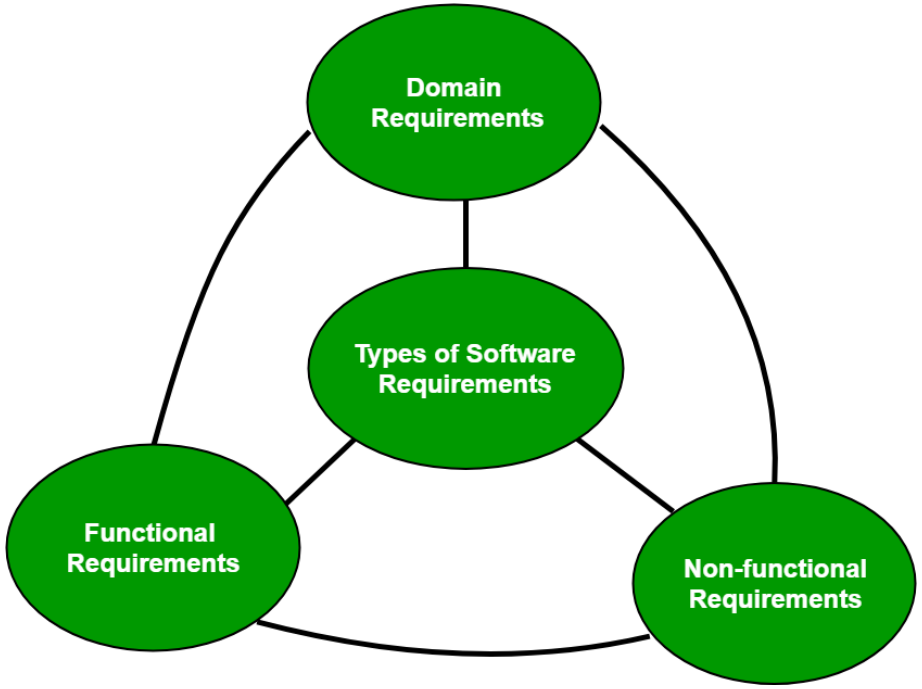
Requirements Analysis

Requirements

- A condition or capability needed by a user to solve a problem or achieve an objective.
- The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

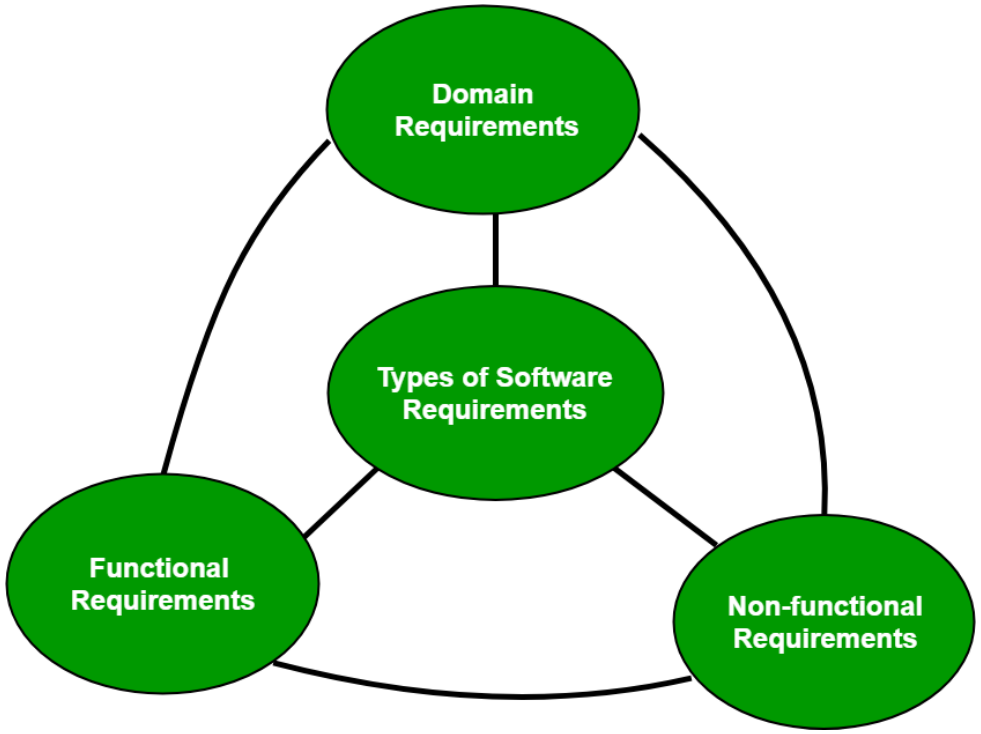
Requirement types

- **Functional Requirements:** Functional requirements describe what the software system should do. They outline the specific functions, features, and interactions that the software must provide to fulfill its intended purpose. These requirements detail the expected behavior of the system from a functional perspective. Examples of functional requirements could include user authentication, data processing, report generation, and any other specific actions or operations the software needs to perform.



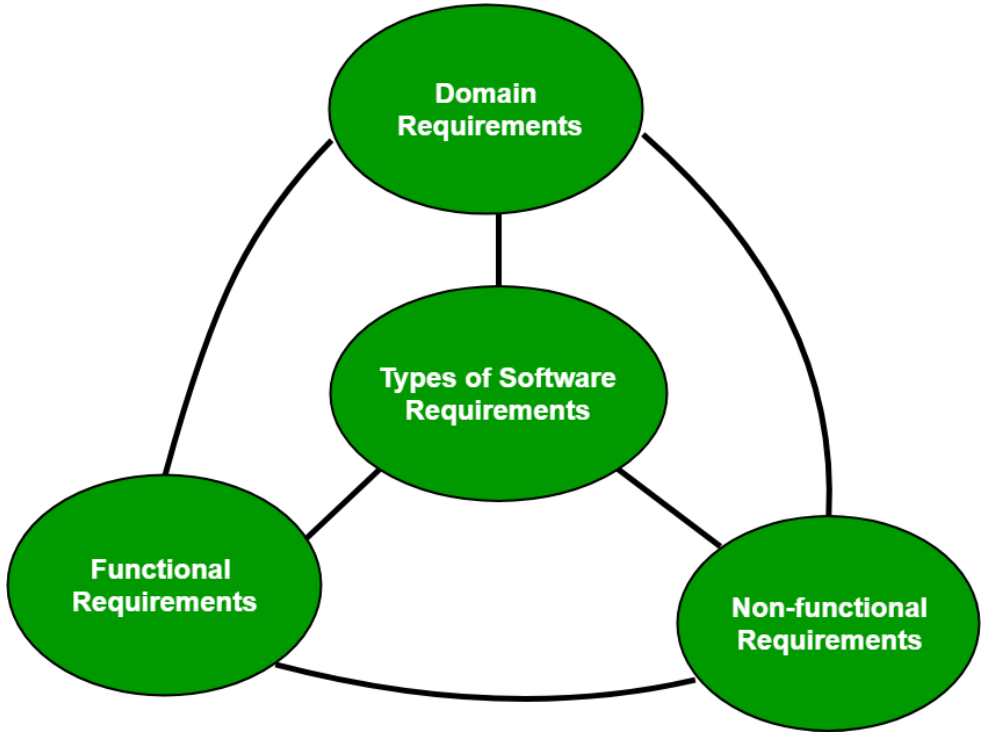
Requirement types

- **Non-Functional Requirements:** Non-functional requirements, also known as quality attributes or constraints, specify the qualities and characteristics that the software system must exhibit. These requirements address attributes such as performance, security, usability, reliability, scalability, and maintainability. Non-functional requirements ensure that the software not only works correctly (functional requirements) but also meets certain standards of performance and usability. Examples of non-functional requirements include response times, data encryption, user interface responsiveness, and system availability.



Requirement types

- **Domain Requirements:** Domain requirements are specific to the domain in which the software will be used. They capture the unique aspects, rules, and constraints related to the industry, business, or application area for which the software is being developed. Domain requirements help ensure that the software aligns with the specific needs and characteristics of the target environment. For example, if developing software for a healthcare domain, domain requirements might include compliance with medical regulations and specific data privacy considerations.



Requirement Management

Requirement Management process includes planning, monitoring, analyzing, communicating and managing of those requirements. If the requirements are not managed well, the end product will get affected adversely. .

The purpose of the Requirement Management is to ensure that business, stakeholders, and solution requirements and designs are aligned with one another.

The Requirement Management–

✓ Begins with the representation of business need as a requirement

✓ Continues through the development of a solution and

✓ Ends when a solution and the requirements that represent it

Why it is important

1: Clear Understanding: Properly managed requirements ensure that all stakeholders have a shared and accurate understanding of what needs to be built. This reduces misunderstandings and miscommunications during development.

2: Project Scope: Effective requirements management helps define the project scope, outlining what will and will not be included in the final product. This prevents scope creep, where additional features are added without proper consideration.

3: Resource Allocation: Well-managed requirements assist in resource allocation, as teams can focus on building features that provide the most value to stakeholders, avoiding wasted efforts on unnecessary functionalities.

Why it is important

4: Risk Mitigation: Thoroughly analyzing and validating requirements can help identify potential risks and challenges early in the project. Addressing these issues in advance can prevent costly rework later on.

5: Quality Assurance: Clearly defined and validated requirements serve as a basis for quality assurance efforts. Testing and validation activities can be aligned with these requirements to ensure the final product meets expectations.

6: Change Control: Requirements lifecycle management provides a structured process for handling changes. Changes can be evaluated for their impact on the project, preventing chaotic and uncontrolled modifications.

Why it is important

7: Communication: Properly managed requirements facilitate communication among stakeholders, development teams, testers, and other project members. Everyone understands the goals and objectives, leading to a more efficient and collaborative environment.

8: Traceability: Traceable requirements establish links between various project artifacts. This helps track the development process, ensuring that each requirement is implemented, tested, and delivered as intended.

9: Customer Satisfaction: When requirements are effectively managed, the end product aligns closely with stakeholder needs and expectations, leading to higher customer satisfaction.

10: Efficient Development: Clear requirements reduce ambiguity during development, enabling teams to work more efficiently and make informed decisions.

Requirement Life Cycle

- Elicitation
- Analysis and Documentation
- Validation and Verification
- Prioritization and Negotiation
- Change Management:
- Communication & Collaboration
- Traceability and Impact Analysis
- Implementation and Testing
- Validation and Acceptance
- Deployment and Maintenance
- Retirement and Archival

Stages

Elicitation: The process begins with the gathering of requirements from stakeholders, users, and other sources. Techniques such as interviews, surveys, and workshops are used to collect information about what the product or system needs to achieve.

Analysis and Documentation: The collected requirements are analyzed to ensure they are clear, consistent, and complete. Ambiguities, contradictions, and gaps are addressed. The requirements are then documented in a structured format, which could include use cases, user stories, functional specifications, and more.

Validation and Verification: Validation ensures that the documented requirements accurately represent the stakeholders' needs and expectations. Verification focuses on checking the quality of the requirements documentation itself, ensuring it adheres to standards and guidelines.

Stages

Prioritization and Negotiation: In cases where requirements conflict or resources are limited, requirements are prioritized based on their importance and feasibility. Stakeholders might negotiate to reach a consensus on the most critical requirements.

Change Management: As the project progresses, requirements may change due to evolving stakeholder needs or market conditions. Changes are managed through a formal change control process to ensure they are evaluated, documented, and approved before implementation.

Communication and Collaboration: Effective communication ensures that all project stakeholders, including developers, designers, testers, and users, understand the requirements and their implications. Collaboration between different teams helps maintain a shared vision of the project.

Stages

Traceability and Impact Analysis: Requirements traceability establishes links between requirements and various project artifacts. This allows for tracking how each requirement is addressed throughout the project. Impact analysis helps assess the effects of changes to requirements on the project as a whole.

Implementation and Testing: The development team translates the requirements into design specifications, code, and other tangible components. Testing activities ensure that the implemented features meet the specified requirements and function as intended.

Validation and Acceptance: After development and testing, the product or system is validated to ensure it meets the initial requirements. Stakeholders participate in acceptance testing to confirm that the product aligns with their needs and expectations.

Stages

Deployment and Maintenance: Once validated and accepted, the product is deployed to its intended environment. Maintenance activities involve addressing issues that arise, making updates based on feedback, and adapting to changing requirements.

Retirement and Archival: In the case of a product becoming obsolete or reaching the end of its lifecycle, the requirements documentation and related artifacts may be archived or retained for future reference or compliance purposes.

Example - Development of a data visualization dashboard for a retail company:

Elicitation:

- Identify stakeholders: Retail managers, marketing analysts, sales teams, and IT staff are stakeholders.
- Conduct interviews: Gather insights from stakeholders on the type of data they need to analyze (sales, inventory, customer behavior, etc.).
- Gather data sources: Identify the various data sources available within the company, such as databases and spreadsheets.
- Document initial requirements: Create a preliminary list of data visualizations and interactive features stakeholders are interested in.

Example - Development of a data visualization dashboard for a retail company:

Analysis and Documentation:

- Define data requirements: Specify the types of data that need to be visualized, such as historical sales, product inventory, and customer demographics.
- Create user stories: Write user stories like "As a retail manager, I want to view monthly sales trends to identify peak seasons."
- Design data flows: Outline how data will be extracted, transformed, and loaded into the dashboard.
- Identify performance expectations: Define response time expectations for loading and interacting with data visualizations.

Example - Development of a data visualization dashboard for a retail company:

Validation and Verification:

- Review data requirements: Ensure that each data visualization aligns with the stakeholders' needs and objectives.
- Validate with stakeholders: Present mockups or prototypes to stakeholders to verify that the proposed visualizations meet their expectations.
- Verify accuracy: Cross-check data requirements against available data sources to ensure accuracy.

Example - Development of a data visualization dashboard for a retail company:

Prioritization and Negotiation:

- Rank visualization types: Assign priorities based on the importance of different visualizations (e.g., sales by region vs. customer segmentation).
- Define MVP visualizations: Identify the core visualizations that will provide the most immediate value to stakeholders.
- Negotiate scope: Discuss scope trade-offs based on available resources and time constraints.

Example - Development of a data visualization dashboard for a retail company:

Change Management:

- Handle new data sources: If stakeholders request additional data sources, evaluate their impact on design and development.
- Assess new visualizations: Consider the feasibility and implications of adding new types of visualizations to the dashboard.

Example - Development of a data visualization dashboard for a retail company:

Communication and Collaboration:

- Collaborate with analysts: Work closely with data analysts to understand their requirements and data transformation needs.
- Engage with developers: Communicate the design requirements, data structures, and interactive features to the development team.

Example - Development of a data visualization dashboard for a retail company:

Traceability and Impact Analysis:

- Establish traceability: Link each visualization requirement to specific data sources and transformation processes.
- Analyze impact of changes: Assess how modifications to data sources or visualization types may affect other parts of the dashboard.

Example - Development of a data visualization dashboard for a retail company:

Implementation and Testing:

- Design data models: Develop database structures and schemas to support the required data visualizations.
- Develop data transformation scripts: Code data processing logic to transform raw data into a format suitable for visualization.
- Build interactive visualizations: Create charts, graphs, and dashboards using data visualization tools.
- Conduct unit testing: Test each visualization and data transformation script individually.

Example - Development of a data visualization dashboard for a retail company:

Validation and Acceptance:

- Conduct integration testing: Test the entire dashboard, ensuring that data is accurately visualized and interactive features work as intended.
- User acceptance testing: Allow stakeholders to use the dashboard to validate that it meets their analytical needs.

Example - Development of a data visualization dashboard for a retail company:

Deployment and Maintenance:

- Deploy the dashboard: Make the dashboard accessible to stakeholders through a secure web portal or application.
- Monitor performance: Continuously monitor dashboard performance and respond to any technical issues or data discrepancies.

Example - Development of a data visualization dashboard for a retail company:

Retirement and Archival:

- Document final state: If a new version of the dashboard is developed, document the final state of this version, including requirements and design documents.
- Archive assets: Store all relevant documentation and artifacts for future reference, compliance, or potential updates.

Assignment

Requirement life cycle for Customer segmentation model for a telecommunications company.

Design Definition

- Design definition refers to the process of creating a detailed plan or blueprint for how a software system will be structured, organized, and implemented. It involves translating the high-level requirements and concepts into specific technical specifications that guide the actual development process. Design definition is a crucial step that bridges the gap between requirement analysis and implementation, providing developers with clear guidance on how to build the software.

Key Aspects

Architectural Design:

1. System architecture: Designing the overall structure of the software system, including the arrangement of modules, components, and their interactions.
2. High-level system components: Defining the major building blocks and their relationships.

Component Design:

1. Detailed module specifications: Designing the individual modules or components identified in the architectural design.
2. Interfaces: Defining the interfaces between modules, specifying how they communicate and exchange data.

Key Aspects

Database Design:

1. Schema design: Creating the database schema, including tables, fields, and relationships.
2. Data integrity: Ensuring that data stored in the database remains consistent and accurate.

User Interface (UI) Design:

1. User interaction flow: Planning the sequence of screens and actions that users will encounter.
2. Visual design: Defining the look and feel of the user interface, including colors, typography, and layout.

Key Aspects

Algorithm Design:

1. Algorithm selection: Identifying the algorithms and data structures that will be used to implement specific functionalities.
2. Pseudocode or flowcharts: Providing a detailed description of how algorithms will execute step by step.

Error Handling and Exception Handling:

1. Error scenarios: Defining how the software should handle unexpected errors, exceptions, and edge cases.
2. Error messages: Designing user-friendly error messages and feedback.

Key Aspects

Security Design:

1. Authentication and authorization: Designing how users will be authenticated and authorized to access different parts of the software.
2. Data encryption: Planning how sensitive data will be stored and transmitted securely.

Performance Optimization:

1. Resource optimization: Designing for efficient memory usage, processing speed, and network bandwidth.
2. Caching strategies: Defining how frequently accessed data will be cached to improve performance.

Key Aspects

Integration and APIs:

1. External integrations: Designing how the software will interact with external systems or services via APIs.
2. API design: Creating clear and consistent APIs for internal components to communicate.

Documentation and Communication:

1. Design documents: Creating detailed documentation that outlines the design decisions, rationale, and technical specifications.
2. Communication with developers: Ensuring that the design is effectively communicated to the development team for implementation.


Q&A
THANK YOU !