SE Module 2

8-Module 2_ Requirements Engineering, UML Model,-10-08-2024

1. The Software Requirement

- **Software Requirements**: Define the features, functions, and expectations of a system. They describe what the software should do from the user's point of view, and they can be obvious or hidden, known or unknown.

2. Types of Requirements

- **User Requirements**: Written in natural language with diagrams, describing the system's services and constraints from the customer’s perspective.
- **System Requirements**: A detailed document specifying the system’s functions, services, and constraints, often used as a contract.

3. Functional vs. Non-Functional Requirements

- **Functional Requirements**: Describe services the system provides, how it should respond to inputs, and how it behaves in different situations. Example: "System should record sales."
    - Types include external interfaces, business rules, data management, legal requirements, etc.
- **Non-Functional Requirements**: Define quality attributes such as performance, security, scalability, and availability. Example: "The system must process 1000 requests per minute."
    - Examples include scalability, capacity, and recoverability.

4. Domain Requirements

- **Domain Requirements**: Expectations specific to a certain industry or application type (e.g., healthcare, finance, or education). These can be either functional or non-functional.

- Example: In an academic system, "Access to the list of faculty members" is a domain requirement.

5. Software Requirements Document (SRD)

- **SRD**: Official document that outlines the user and system requirements. It's a description of "what" the system should do, not "how."
- **Important**: The SRD is not a design document, but a specification.

6. Requirement Engineering

- **Requirement Engineering**: The process of gathering, analyzing, and documenting the software requirements from the client.
- The goal is to develop a detailed **System Requirements Specification** document that defines what the system will do.

7. Requirement Engineering Process

- **Feasibility Study**: Analyzes whether the project can be achieved in terms of technical, operational, and economic feasibility. Outcomes:
  - **Go ahead**: Proceed with the project.
  - **Do not go ahead**: Stop the project.
  - **Think again**: Reconsider and analyze further.
- **Requirement Gathering**: Collects requirements from the user. This includes business requirements (functional requirements) and user expectations.
  - Tools: Context diagrams, mind maps, use cases, AS-IS and TO-BE models.
- **Steps in Requirement Engineering**:
  1. **Inception**: Define the scope and nature of the system.
  2. **Elicitation**: Gather requirements from users.
  3. **Elaboration**: Refine the gathered requirements.
  4. **Negotiation**: Resolve conflicts and prioritize requirements.
  5. **Specification**: Compile the gathered requirements into a final document.

6. **Validation**: Check if requirements are clear, complete, and consistent.
7. **Management**: Handle changes to requirements throughout the project.

8. Tools and Methods for Gathering Requirements

- **Tools**: Context diagrams, mind maps, use cases, and modeling the "AS-IS" (current state) and "TO-BE" (desired future state) systems.

9. Key Terms

- **Functional Requirements**: What the system should do (e.g., process transactions, record data).
- **Non-Functional Requirements**: Attributes of the system (e.g., response time, security).
- **Domain Requirements**: Industry-specific requirements (e.g., medical, military).
- **Feasibility Study**: Determines if the project is technically and financially viable.

9-Module 2_ Requirements Engineering, UML Model,-10-08-2024

Requirements Engineering

Steps in Requirements Engineering:

1. **Inception**: This is the beginning phase where the project's scope and objectives are defined. Key questions like who is requesting the software, who will use it, and what benefits it will provide are considered. Stakeholders such as customers, end-users, product managers, and engineers are identified.
2. **Elicitation**: The process of gathering requirements from stakeholders begins. A facilitator leads discussions to identify problems and propose solutions. Quality Function Deployment (QFD) may be used to translate customer needs into technical requirements, including value analysis and task deployment.
3. **Elaboration**: The requirements gathered are expanded upon to build an analysis model, incorporating functional and use-case scenarios. Various

diagrams, such as state diagrams, flow-oriented models, and class-based models, may be used to detail the system's structure.

4. **Negotiation**: Stakeholders' needs are prioritized, and conflicts between requirements are resolved. This step involves negotiating the requirements to ensure a "win-win" outcome.
5. **Specification**: The finalized requirements are documented in a clear and concise manner, forming the Software Requirements Specification (SRS).
6. **Validation**: This step ensures that the requirements are correct, complete, and achievable. Various validation techniques are used to ensure that the SRS meets the customer's needs and can be implemented within given constraints.
7. **Management**: This phase involves managing changes to the requirements throughout the project lifecycle.

Detailed Breakdown of Requirements Engineering Steps:

- **Inception**: Focuses on identifying the problem, determining who will benefit from the software, and defining stakeholder roles.
- **Elicitation**: Facilitated meetings are used to gather an initial set of requirements. Techniques such as QFD help translate customer needs into technical requirements.
- **Elaboration**: Develops a detailed analysis model, often using use-cases, state diagrams, and other tools to describe the system's behavior and structure.
- **Negotiation**: Conflicting requirements are resolved, and priorities are set to reach an agreement that satisfies all stakeholders.
- **Specification**: The requirements are written into a formal document, the SRS, detailing what the system should do, without delving into how it will be done.
- **Validation**: Ensures that the requirements meet customer expectations, are complete, consistent, and testable.
- **Management**: Requirements are tracked, and changes are managed effectively throughout the project's lifecycle.

Key Concepts and Tools Used in Requirements Engineering:

1. **Use Cases**: Descriptions of how users interact with the system to achieve goals. Each use case includes actors (users), the system, and the goal.
2. **Class Diagrams**: Represent the static structure of the system, including classes, attributes, and operations. They describe the responsibilities and relationships between system components.
3. **State Diagrams**: Depict the behavior of a system as it transitions between various states over time, helping to model dynamic changes in the system.
4. **Data Flow Diagrams (DFD)**: Illustrate the flow of data between processes and external entities. DFDs are categorized into levels (Level 0, 1, 2, etc.) to represent different layers of abstraction.
5. **Activity Diagrams**: Represent the workflow of activities within the system. They are useful for modeling sequential processes and decision points.
6. **Sequence Diagrams**: Focus on the order of interactions between system components over time. They show how objects communicate in terms of messages.
7. **Navigation Modeling**: Defines how users navigate through a system, considering the most efficient paths and how to handle errors.

Requirements Validation:

1. **Checking for Validity, Consistency, and Completeness**: Ensures that the requirements meet the user's needs, are free from conflicts, and include all necessary functionality.
2. **Realism and Verifiability**: Ensures that the requirements can be implemented within the given constraints and are testable.
3. **Validation Techniques**: Includes requirements reviews, prototyping, and test-case generation to ensure the requirements are practical and accurate.

Requirements Modeling for Web Applications:

1. **Content Analysis**: Identifies the various content elements of a web application (text, images, audio, etc.), using data modeling to describe each content object.

2. **Interaction Analysis**: Describes the user interactions with the system, typically through use cases and sequence diagrams.
3. **Functional Analysis**: Describes the operations that the web application will perform on content, detailing all functions and processing steps.
4. **Configuration Analysis**: Identifies the technical environment where the web application will operate, including server and client-side configurations.

Key Points in Requirements Modeling for Web Applications:

- **Scenario-based Models**: Describes the system from the perspective of various actors (users, systems).
- **Class-oriented Models**: Focus on the structure of objects (attributes and operations) and their relationships.
- **Flow-oriented Models**: Detail how data is processed and moved through the system.
- **Behavioral Models**: Depict how the system reacts to external events.

Conclusion:

This content provides an in-depth look at the steps and tools used in **requirements engineering**, including how to gather, specify, and validate software requirements. It also covers specific techniques such as use-case analysis, class diagrams, and flow-oriented modeling, particularly for web applications. The process ensures that the final software product aligns with stakeholders' expectations and can be successfully developed and deployed.

10-Developing Use Cases, Building the Requirements Model-13-08-2024

1. Software and Software Engineering

- **Software**: A collection of programs, data, and instructions that perform tasks on a computer.
- **Software Engineering**: The discipline of designing, developing, testing, and maintaining software systems efficiently and effectively.

2. The Software Process

- The software process includes all activities from initial concept to the final product (e.g., planning, design, coding, testing, and maintenance).
- Key models: Waterfall, Iterative, Incremental, Agile, and V-Model.

3. Agile Process

- Agile is an iterative, incremental approach that focuses on flexibility, collaboration, and customer feedback.
- **Key principles**:
  - Deliver working software frequently.
  - Customer collaboration over contract negotiation.
  - Responding to change over following a fixed plan.

4. Principles that Guide Practice

- **Simplicity**: Focus on the essential parts of the system.
- **Communication**: Keep stakeholders involved and well-informed.
- **Feedback**: Regular feedback to improve the product.
- **Adaptability**: Ability to adapt to changing requirements.

---

5. Requirement Engineering

- **Requirement Engineering**: The process of gathering, analyzing, specifying, validating, and managing the requirements for a system.
- It involves 7 tasks:
  1. **Inception**: Understanding the problem.
  2. **Elicitation**: Gathering requirements from stakeholders.
  3. **Elaboration**: Clarifying and refining requirements.
  4. **Negotiation**: Resolving conflicts between stakeholders.
  5. **Specification**: Writing down clear and unambiguous requirements.

6. **Validation**: Ensuring requirements are correct and feasible.
7. **Management**: Keeping track of changing requirements.

---

6. Establishing the Groundwork

- Identify **stakeholders** (people who influence the project).
- Recognize **multiple viewpoints** to understand different perspectives.
- **Collaborate** to build a shared understanding.
- Ask key questions to clarify goals.

7. Eliciting Requirements

- **Collaborative Requirements Gathering**: Meetings where both software engineers and stakeholders discuss the system's needs.
- **Quality Function Deployment (QFD)**: Prioritize requirements into:
  1. **Normal**: Expected but not essential.
  2. **Expected**: Things customers expect.
  3. **Exciting**: Extra features that delight customers.
- **Usage Scenarios**: Real-life examples of how the system will be used.

8. Developing Use Cases

- **Use Cases** describe how users interact with the system.
  - **Actors**: People or devices that interact with the system.
  - **Goal**: What the actor wants to achieve from the interaction.
  - **Preconditions**: Requirements that must be met before the use case begins.
  - **Exceptions**: What happens if something goes wrong.
  - **Variations**: Different ways the use case could play out.

9. Building the Requirements Model

- Create a model that includes:

- **Scenario-based** elements: Descriptions of how users interact with the system.
- **Class-based** elements: The system's structure (e.g., objects or data).
- **Behavioral** elements: How the system responds to events.

10. Negotiating Requirements

- **Negotiation** helps reconcile conflicting stakeholder goals.
- Steps:
  1. Identify key stakeholders.
  2. Understand their goals.
  3. Find a “win-win” solution.

11. Validating Requirements

- Check that each requirement is:
  - **Consistent**: Doesn't contradict others.
  - **Feasible**: Possible to implement in the technical environment.
  - **Testable**: Can be verified once implemented.
  - **Necessary**: Important for the system's objectives.
  - **Unambiguous**: Clear and easy to understand.
  - **Attributable**: Can be traced back to a stakeholder.

11-Design within the Context of Software Engineering,-20-08-2024

Software Design Overview

- **Software Design** is the process of planning and creating software systems. It’s crucial to software engineering as it helps translate requirements into a working system.
- **Goals of Software Design**:

- **Firmness**: No bugs.
- **Commodity**: Software must meet its intended purpose.
- **Delight**: The experience should be pleasant for users.

Design Process

- **Iterative Process**: Design evolves and refines over time.
- **Key Stages**:
  1. **Data/Class Design**: Converts class models into real data structures.
  2. **Architectural Design**: Defines software structure and major components.
  3. **Interface Design**: Defines how software communicates with users and other systems.
  4. **Component-Level Design**: Details how software components interact.

Design Quality Guidelines

- A good design must:
  - **Implement all requirements**.
  - Be **readable** and **understandable** for developers.
  - Provide a **complete picture** of the software.

Design Quality Attributes

- **Functionality**: Features and security.
- **Usability**: User-friendliness.
- **Reliability**: Consistency and error-free operation.
- **Performance**: Speed, resource use, and efficiency.
- **Supportability**: Ease of maintenance and adaptability.

Design Concepts

1. **Abstraction**: Simplifies complex systems by focusing on relevant details.
2. **Architecture**: Defines the system's structure.

3. **Patterns**: Reusable design solutions for common problems.
4. **Separation of Concerns**: Divide complex problems into manageable parts.
5. **Modularity**: Break the system into smaller, manageable modules.
6. **Information Hiding**: Hide the details of modules to reduce changes when modifying the system.
7. **Functional Independence**: Each module should perform a single task independently to reduce interdependencies.
8. **Refinement**: Start with high-level descriptions and gradually add details.
9. **Aspects**: Address concerns that affect multiple parts of the system.
10. **Refactoring**: Improve the internal structure without changing the external behavior.
11. **OO Design Concepts**: Includes inheritance, polymorphism, and messages.
12. **Design Classes**: Types of classes like user interface, business domain, and process classes that make up the system.

Conclusion

Software design is central to building robust, maintainable systems. It requires understanding the requirements and translating them into structured, efficient, and user-friendly solutions. Key design principles like abstraction, modularity, and functional independence help manage complexity and improve the quality of the software.

12-Design Model.-24-08-2024

Design Model Overview

- The **Design Model** evolves in two ways:
    1. **Process Dimension**: The design grows as tasks are completed.
    2. **Abstraction Dimension**: Design details are refined from high-level analysis to specific design.

Main Design Elements

1. **Data Design**:
    - Focuses on organizing and modeling data.
    - Starts with a high-level view (how users see it) and becomes more technical for implementation.
2. **Architectural Design**:
    - Provides an overall system structure, often shown as interconnected subsystems.
    - Derived from the application domain and requirements, using patterns for structure.
3. **Interface Design**:
    - Defines how information flows in and out of the system.
    - Includes user interfaces, external system connections, and internal component communication.
4. **Component-Level Design**:
    - Breaks down the system into detailed parts (like rooms in a house), focusing on internal details of each software component.
    - In object-oriented design, it's represented with UML diagrams to show processing logic and interactions.
5. **Deployment-Level Design**:
    - Shows how the software will be allocated to physical hardware (computing environment).
    - A deployment diagram illustrates how the software components and interfaces fit into the physical system.

Conclusion

The design model is a structured approach that refines system elements from high-level abstractions to detailed implementations, covering data, architecture, components, interfaces, and deployment for a complete software solution.