EC3290 – Software Requirements Engineering

Topic 1 – Introduction to Requirements Engineering

CLO 1

4 Hours

Introduction to Software Engineering

Definition:

Software engineering is the application of engineering principles to the development, operation, and maintenance of software systems. It involves a systematic, disciplined, and quantifiable approach to software development, ensuring high quality and efficiency.

Key Components:

1. **Processes:**
   These are structured sequences of activities that guide software development, such as planning, analysis, design, implementation, testing, and maintenance. Processes ensure a systematic approach and are often tailored to project needs.

2. **Tools:**
   Software tools are used to support processes and methodologies. Examples include integrated development environments (IDEs), version control systems, and testing frameworks.

3. **Methodologies:**
   These are frameworks that dictate how processes are implemented. Examples include Agile, Waterfall, and DevOps. Methodologies help teams align on goals and strategies throughout the project lifecycle.

Software Development Lifecycle (SDLC) Overview

Planning

Analysis

Design

Implementation

Maintenance

Importance of Processes in Software Development

- **Ensures Consistency and Quality:**
  Well-defined processes help maintain consistency across teams and projects. By following standard procedures, teams can deliver software that meets quality expectations and adheres to best practices.
- **Manages Complexity and Reduces Risks:**
  Software development often involves handling complex systems and technologies. Processes help break down these complexities into manageable steps and identify potential risks early, allowing teams to mitigate them proactively.
- **Facilitates Communication Among Stakeholders:**
  Processes provide a common framework and terminology for developers, project managers, clients, and other stakeholders. This fosters better collaboration, clearer expectations, and alignment on project goals.

Relationship Between Software Processes and Requirements

- **Foundation of Subsequent Phases:**
  Requirements serve as the cornerstone of software development. They define what the software must do and set the scope of the project. All subsequent phases—design, implementation, testing, and maintenance—rely on accurate and comprehensive requirements.
- **Impact of Poor Requirements:**
  Ambiguous, incomplete, or incorrect requirements can lead to misaligned development efforts, increased costs, and delays. In worst-case scenarios, poor requirements can result in the complete failure of a project.
- **Processes to Improve Requirement Quality:**
  - **Requirement Elicitation:** Engaging stakeholders through interviews, surveys, and workshops to gather detailed requirements.
  - **Requirement Validation:** Ensuring requirements are clear, consistent, and feasible before moving to the design phase.
  - **Requirement Management:** Continuously tracking and updating requirements throughout the project lifecycle.

Definition of Requirements Engineering

- **Definition:**
  Requirements engineering is the systematic process of identifying, documenting, and maintaining software requirements. It ensures that the software being developed aligns with user needs and business goals. This process is iterative and involves continuous interaction with stakeholders.

- **Key Activities in Requirements Engineering:**

  1. **Elicitation:**
     Gathering requirements from stakeholders through interviews, surveys, observation, and workshops.
  2. **Documentation:**
     Structuring the gathered requirements into clear, unambiguous, and verifiable documents, such as Software Requirement Specifications (SRS).
  3. **Validation:**
     Ensuring that requirements are complete, feasible, and correctly understood by all parties.
  4. **Management:**
     Maintaining and updating requirements throughout the software development lifecycle to address changes and evolving needs.

Role of Requirements Engineering

- **Bridges the Gap Between Stakeholders and Developers:**
  Requirements engineering acts as a mediator to translate stakeholder needs into technical specifications. It ensures that developers understand what the stakeholders want and why it's important.
- **Establishes a Clear Understanding of Project Goals:**
  By defining and documenting requirements upfront, all project participants—stakeholders, developers, testers, and managers—gain a shared understanding of the project's objectives and scope.
- **Supports Decision-Making:**
  Well-defined requirements provide a basis for making informed decisions during the project, such as prioritizing features or resolving conflicts between stakeholder expectations.
- **Enhances Collaboration:**
  It creates a structured process for continuous communication between the technical and non-technical members of the project team.

Importance of Requirements Engineering

- **Prevents Scope Creep:**
  Requirements engineering helps in clearly defining the project scope, preventing uncontrolled expansion of features or functionality during development. This ensures the project stays on track and within budget.
- **Reduces Development Costs and Time:**
  Addressing and refining requirements early in the project lifecycle helps identify potential issues and misunderstandings, reducing costly rework during later stages.
- **Improves Software Quality and User Satisfaction:**
  By focusing on user needs and expectations, requirements engineering ensures that the delivered software is both functional and user-friendly, leading to higher satisfaction and usability.
- **Minimizes Risks:**
  A clear understanding of requirements helps identify risks early, such as technical feasibility or conflicts between stakeholder priorities, enabling better risk management strategies.
- **Ensures Compliance:**
  For industries with regulatory requirements, the process ensures that all necessary standards are met.

Challenges in Requirements Engineering

- **Changing Requirements:**
  Stakeholders' needs may evolve over time due to market changes, new business goals, or feedback from early stages of development. Managing these changes while minimizing their impact is a significant challenge.
- **Ambiguities and Misunderstandings:**
  Requirements may be written or communicated in a way that is unclear, leading to differing interpretations by stakeholders and developers.
- **Communication Gaps Among Stakeholders:**
  Stakeholders may have conflicting goals or use different terminology, making it challenging to align their expectations.
- **Limited Stakeholder Involvement:**
  If stakeholders are not actively engaged in the requirements process, critical needs may be overlooked or misunderstood.
- **Technical Feasibility Issues:**
  Some requirements may not be technically or financially feasible, requiring negotiation and adjustment to align with constraints.

Challenges in Requirements Engineering

- **Addressing These Challenges:**
  - Use clear, unambiguous language and visual aids like diagrams.
  - Involve stakeholders continuously throughout the process.
  - Employ requirement management tools for tracking changes.
  - Conduct regular validation sessions to ensure alignment among all parties.

Overview of Requirements Engineering Activities

- **Requirements Engineering Activities:**
  1. **Elicitation:**
     The process of gathering requirements from stakeholders to understand what the system should do.
  2. **Analysis:**
     Understanding, refining, and prioritizing requirements to ensure they are feasible and align with project goals.
  3. **Specification:**
     Documenting requirements in a structured format to serve as a reference for all project phases.
  4. **Validation:**
     Verifying that the documented requirements meet user needs and are complete, consistent, and feasible.
  5. **Management:**
     Handling changes to requirements throughout the project lifecycle to ensure alignment with evolving needs.

Elicitation

- **Definition:**
  Elicitation is the process of gathering and discovering requirements from stakeholders and other sources. It is the foundation of requirements engineering and involves active engagement with stakeholders.
- **Techniques:**
  1. **Interviews:**
     Conducting one-on-one or group discussions to understand stakeholder needs and expectations.
  2. **Surveys and Questionnaires:**
     Distributing structured forms to collect information from a larger group of stakeholders.
  3. **Brainstorming Sessions:**
     Facilitating collaborative sessions to generate ideas and uncover hidden needs.
  4. **Document Analysis:**
     Reviewing existing documentation, such as business plans or system

Analysis

- **Definition:**
  Analysis involves understanding, refining, and organizing elicited requirements to ensure they are feasible, consistent, and aligned with project goals.
- **Key Activities:**
  1. **Identifying Dependencies:**
     Understanding how different requirements are related and ensuring no conflicts exist between them.
  2. **Prioritizing Requirements:**
     Categorizing requirements based on their importance and urgency to stakeholders. Techniques like the MoSCoW method (Must have, Should have, Could have, Won't have) are often used.
  3. **Feasibility Assessment:**
     Ensuring that the requirements can be achieved within the project's technical, financial, and time constraints.

MoSCoW Method – Must Have

- **Definition:** These are the requirements that are critical to the success of the project. Without them, the system cannot function, and the project would be considered a failure.
- **Characteristics:**
  - Essential for achieving business goals or meeting legal and regulatory requirements.
  - Directly impacts the system's core functionality.
- **Examples:**
  - A banking app **must have** secure login functionality.
  - An e-commerce website **must have** a shopping cart feature.

MoSCoW Method – Should Have

- **Definition:** These are important requirements but not critical for immediate success. They are often considered high-priority but not mandatory for the first release.
- **Characteristics:**
  - Adds significant value but can be deferred if necessary.
  - May be included if time and resources permit.
- **Examples:**
  - A system **should have** multiple language support.
  - A project management tool **should have** integration with other tools like Slack or Teams.

MoSCoW Method – Could Have

- **Definition:** These are desirable but not essential requirements. They have a lower priority and are often included if time, budget, and resources allow.
- **Characteristics:**
  - Nice-to-have features that enhance user experience.
  - Does not impact core functionality or business goals.
- **Examples:**
  - A mobile app **could have** a dark mode.
  - A website **could have** animated transitions between pages.

MoSCoW Method – Won’t Have

- **Definition:** These are requirements that are agreed to be out of scope for the current project or release but may be considered for future iterations.
- **Characteristics:**
  - Explicitly documented to avoid confusion or scope creep.
  - Can be revisited in later phases or versions.
- **Examples:**
  - A system **won’t have** voice recognition in the initial release.
  - A website **won’t have** augmented reality features in Phase 1.

Specification

- **Definition:**
  Specification involves documenting the requirements in a clear, structured, and unambiguous format that serves as a reference for all project stakeholders.
- **Examples of Documentation Formats:**
  1. **Use Case Diagrams:**
     Visual representations of interactions between users and the system to achieve specific goals.
  2. **User Stories:**
     Short, simple descriptions of a feature from the perspective of the end user.
  3. **Software Requirements Specification (SRS):**
     A detailed document that outlines functional and non-functional requirements, system constraints, and other relevant details.

Validation

- **Definition:**
  Validation ensures that the documented requirements meet user needs and expectations and are complete, consistent, and feasible.
- **Techniques:**
  1. **Reviews and Inspections:**
     Conducting walkthroughs and reviews with stakeholders and team members to verify requirements.
  2. **Prototyping:**
     Creating a preliminary version of the system to demonstrate functionality and gather feedback.
  3. **Test Cases:**
     Writing test cases based on requirements to verify their completeness and correctness during development.

Management

- **Definition:**
  Management involves handling changes to requirements throughout the project lifecycle, ensuring that updates align with stakeholder needs and do not disrupt the project.
- **Tools and Techniques:**
  1. **Version Control Systems:**
     Tools like Git to track changes to requirement documents and maintain version history.
  2. **Requirements Management Software:**
     Tools like Jira or IBM Rational DOORS to organize, prioritize, and track requirements changes.
  3. **Change Control Process:**
     A formal process for evaluating, approving, and documenting requirement changes to ensure traceability.

Introduction to Types of Requirements

- **Overview:**
  Requirements are the foundation of software development, guiding the design, implementation, and validation of a system. They can be broadly classified into two main categories:

  1. **Functional Requirements:**
     Define the specific functionalities or features the system must provide.
  2. **Non-Functional Requirements:**
     Define the quality attributes or performance criteria the system must meet.

Functional Requirements

- Functional requirements specify what the system should do to meet user needs and achieve business goals. They describe the behavior of the system in response to inputs and interactions.

- **Key Characteristics:**
  - Directly related to user interactions.
  - Clearly define inputs, processes, and outputs.
  - Serve as the basis for system design and testing.

Functional Requirements

- **Examples:**
  1. **Login Functionality:**
     The system should allow users to log in with a username and password.
  2. **Search Feature:**
     The system should provide a search bar that returns relevant results based on user queries.
  3. **Report Generation:**
     The system should generate monthly sales reports in PDF format.

Non-Functional Requirements

- Non-functional requirements describe how the system should perform rather than what it should do. These requirements focus on quality attributes, ensuring the system operates efficiently and meets user expectations.

- **Key Characteristics:**
  - Address operational and environmental aspects.
  - Impact user satisfaction and system reliability.
  - Often harder to measure and validate.

Non-Functional Requirements

- **Examples:**
  1. **Performance:**
     The system should handle 1,000 concurrent users without performance degradation.
  2. **Security:**
     The system should encrypt user data using AES-256 encryption.
  3. **Scalability:**
     The system should support increasing workloads by adding additional servers.
  4. **Usability:**
     The system should have an intuitive user interface with accessible navigation.

Comparison: Functional vs. Non-Functional Requirements

Aspect	Functional Requirements	Non-Functional Requirements
Definition	What the system should do	How the system should perform
Focus	Specific functionality and tasks	Quality attributes and performance
Examples	Login, search, report generation	Performance, security, scalability, usability
Measurement	Measured through system functionality	Measured through system benchmarks
Impact	Directly impacts system operations	Impacts user satisfaction and system efficiency

Importance of Clear Requirements

- **Functional Requirements:**
  - Clearly defined functional requirements ensure the system fulfills user tasks effectively.
  - They reduce ambiguity and guide the design and implementation processes.
  - Example: If the login functionality is unclear, users may experience frustration or errors.
- **Non-Functional Requirements:**
  - These requirements ensure the system performs well under real-world conditions.
  - They directly impact user satisfaction, reliability, and scalability.
  - Example: A slow or unresponsive system can lead to poor user

What Makes a Good Requirement?

- **Characteristics of a Good Requirement:**
  1. **Clear:**
     The requirement should be easy to understand and leave no room for misinterpretation.
     1. Example: "The system shall generate a weekly sales report in PDF format."
  2. **Concise:**
     The requirement should be brief and to the point while still providing enough detail.
  3. **Testable:**
     The requirement must be verifiable through testing or demonstration.
     1. Example: "The system shall encrypt all user passwords using AES-256 encryption."
  4. **Consistent:**
     There should be no contradictions between requirements or other project documents.
  5. **Feasible:**
     The requirement should be achievable within the project's constraints of time, budget, and technology.

Common Pitfalls in Writing Requirements

- **Ambiguity:**
  Requirements that can be interpreted in multiple ways lead to confusion.
- Example of a poor requirement: "The system should be fast."
- **Vagueness:**
  Requirements lacking specific details are difficult to implement and test.
- Example: "The system should handle many users."
- **Lack of Detail:**
  Insufficient information leaves developers guessing about what is needed.
- **Over-Specification:**
  Overly detailed requirements can limit flexibility and creativity in implementation.

Using Templates and Standards

- **Ensures Uniformity:**
  All requirements are documented in a consistent format, making them easier to understand and review.
- **Promotes Completeness:**
  Templates guide writers to include all necessary information, reducing the risk of missed details.
- **Enhances Quality:**
  Standards ensure that requirements meet industry best practices.

Examples of Well-Written Requirements

- **Functional Requirement:**
  - **Good Example:** "The system shall allow users to reset their password using their registered email address."
  - **Why it's good:** It's clear, specific, and testable.
- **Non-Functional Requirement:**
  - **Good Example:** "The system shall support up to 100 concurrent users with a response time of less than 2 seconds for any operation."
  - **Why it's good:** It defines measurable performance criteria.

Techniques to Improve Requirement Quality

- **Use Simple Language:**
  Write requirements in plain, non-technical language that all stakeholders can understand.
- Example: Instead of "implement an asymmetric cryptographic algorithm," write "use public-private key encryption."
- **Avoid Technical Jargon:**
  Use terminology that is accessible to both technical and non-technical stakeholders.
- **Engage Stakeholders in Reviews:**
  Regularly review requirements with stakeholders to ensure alignment and avoid miscommunication.
- **Utilize Visual Aids:**
  Incorporate diagrams, use case models, or flowcharts to supplement textual requirements.
- **Iterative Refinement:**
  Continuously refine requirements based on stakeholder feedback and

"Great software isn’t magic—it’s built on clear goals, solid plans, and knowing what matters most. Nail the basics, and the rest will flow."

End of Topic 1