Lecture 7: Requirements Gathering and Analysis

From User Needs to Software Specifications

State University of Zanzibar (SUZA)
BSc Computer Science

Contents

1 Introduction to Requirements Engineering

1.1 What are Requirements?

Requirements are statements that describe what a software system must do and how it must perform. They form the foundation of any software project.

Why Requirements Matter:

- 40-60% of project failures are due to poor requirements
- Fixing requirements errors late in development costs 100x more
- Clear requirements lead to better estimates and planning
- Requirements are the contract between developers and stakeholders

1.2 Types of Requirements

1.2.1 Functional Requirements

Describe **what** the system should do.

Examples:

- The system shall allow users to register with email and password
- The system shall send email notifications when orders are placed
- Users shall be able to search products by name or category
- The admin shall be able to generate monthly sales reports

1.2.2 Non-Functional Requirements

Describe **how** the system should perform.

Categories:

- **Performance:** Response time, throughput, capacity
- **Security:** Authentication, authorization, encryption
- **Usability:** Ease of use, accessibility, learning curve
- **Reliability:** Uptime, fault tolerance, recovery
- **Scalability:** Handle growth in users and data
- **Maintainability:** Ease of updates and fixes

Examples:

- The system shall respond to user requests within 3 seconds
- The system shall be available 99.9% of the time
- The system shall support 1000 concurrent users
- All passwords shall be encrypted using bcrypt

2 Requirements Gathering Techniques

2.1 1. Stakeholder Interviews

What: One-on-one conversations with key stakeholders

How to Conduct:

1. Prepare questions in advance
2. Start with open-ended questions
3. Listen actively and take notes
4. Ask follow-up questions
5. Summarize and confirm understanding

Sample Questions:

- What problem are you trying to solve?
- Who will use this system?
- What tasks do you do daily that this system should support?
- What are your biggest pain points with the current process?
- What would success look like for this project?

2.2 2. Questionnaires and Surveys

Best For: Gathering input from many users

Tips:

- Keep questions clear and concise
- Use a mix of open and closed questions
- Include rating scales (1-5)
- Test the survey before distributing

2.3 3. Observation

What: Watch users perform their current tasks

Benefits:

- Understand actual workflow (not just described)
- Identify pain points users don't mention
- See workarounds and inefficiencies

2.4 4. Document Analysis

What: Review existing documentation
Documents to Review:

- Current system documentation
- Business process documents
- Reports and forms
- Competitor products

2.5 5. Workshops and Brainstorming

What: Group sessions with stakeholders
Techniques:

- Brainstorming sessions
- JAD (Joint Application Development)
- Focus groups
- Prototyping sessions

3 User Stories

3.1 What is a User Story?

A user story is a short, simple description of a feature from the user's perspective.

3.2 User Story Format

As a [type of user], **I want** [some goal/action], **So that** [benefit/reason].

3.3 Examples

E-commerce System:

- As a **customer**, I want to **search for products by name** so that I can **quickly find what I'm looking for**.
- As a **customer**, I want to **add items to my cart** so that I can **purchase multiple items at once**.
- As an **admin**, I want to **view sales reports** so that I can **track business performance**.

Student Management System:

- As a **student**, I want to **view my grades online** so that I can **track my academic progress**.
- As a **lecturer**, I want to **upload course materials** so that **students can access them anytime**.
- As a **registrar**, I want to **generate transcripts** so that **students can apply for jobs or further studies**.

3.4 INVEST Criteria for Good User Stories

- **I**ndependent - Can be developed separately
- **N**egotiable - Details can be discussed
- **V**aluable - Provides value to user
- **E**stimable - Can estimate effort
- **S**mall - Fits in one sprint
- **T**estable - Can verify completion

3.5 Acceptance Criteria

Each user story should have acceptance criteria that define when it's complete.

Format: Given-When-Then

```
Given [some context/precondition]
When [action is performed]
Then [expected result]
```

Example:

```
User Story: As a customer, I want to login
so that I can access my account.

Acceptance Criteria:
1. Given I am on the login page
   When I enter valid credentials
   Then I am redirected to my dashboard

2. Given I am on the login page
   When I enter invalid credentials
   Then I see an error message

3. Given I am logged in
   When I am inactive for 30 minutes
   Then I am automatically logged out
```

4 Use Cases

4.1 What is a Use Case?

A use case describes a specific interaction between a user (actor) and the system to achieve a goal.

4.2 Use Case Components

Component	Description
Use Case ID	Unique identifier (e.g., UC-001)
Name	Descriptive name (e.g., "User Login")
Actor(s)	Who initiates the use case
Description	Brief summary
Preconditions	What must be true before
Main Flow	Step-by-step normal scenario
Alternative Flows	Variations from main flow
Exception Flows	Error handling
Postconditions	What is true after

4.3 Example Use Case

Use Case ID	UC-001
Name	User Registration
Actor	New User
Description	User creates a new account
Preconditions	User is not logged in
Main Flow 1. User clicks "Register" 2. System displays registration form 3. User enters name, email, password 4. User clicks "Submit" 5. System validates input 6. System creates account 7. System sends confirmation email 8. System displays success message	
Alternative Flow 5a. Email already exists: System displays error message User can try different email	
Postconditions	User account is created and active

5 Requirements Prioritization

5.1 MoSCoW Method

- **Must Have (M):** Critical for launch, non-negotiable
- **Should Have (S):** Important but not critical

- **Could Have (C):** Nice to have if time permits
- **Won't Have (W):** Out of scope for current release

5.2 Example Prioritization

E-commerce Project:

- **Must Have:**
  - User registration and login
  - Product listing
  - Shopping cart
  - Checkout process
- **Should Have:**
  - Product search
  - Order history
  - Email notifications
- **Could Have:**
  - Product reviews
  - Wishlist
  - Social media login
- **Won't Have:**
  - Mobile app
  - AI recommendations

6 Software Requirements Specification (SRS)

6.1 SRS Document Structure

1. **Introduction**
   - Purpose
   - Scope
   - Definitions and acronyms
   - References
2. **Overall Description**
   - Product perspective
   - Product features
   - User classes

- Operating environment
- Constraints
- Assumptions

3. **Functional Requirements**
   - Detailed requirement descriptions
   - User stories
   - Use cases
4. **Non-Functional Requirements**
   - Performance
   - Security
   - Usability
   - Reliability
5. **Appendices**
   - Glossary
   - Diagrams
   - Mockups

7 Requirements Validation

7.1 Validation Techniques

- **Reviews:** Stakeholders review requirements document
- **Prototyping:** Build mockups to validate understanding
- **Test Case Generation:** Write tests to verify requirements are testable
- **Traceability:** Ensure all requirements are linked to tests

7.2 Good Requirements Characteristics (SMART)

- **S**pecific - Clear and unambiguous
- **M**easurable - Can be verified
- **A**chievable - Technically feasible
- **R**elevant - Aligned with project goals
- **T**ime-bound - Has deadline or priority

8 Common Mistakes to Avoid

1. **Vague requirements:** "The system should be fast" (How fast?)
2. **Missing stakeholders:** Not consulting all user types
3. **Gold plating:** Adding unnecessary features
4. **No prioritization:** Treating all requirements equally
5. **Lack of validation:** Not confirming requirements with stakeholders

9 Practical Exercise

For your project, complete the following:

1. Identify all stakeholders/user types
2. Write at least 10 user stories
3. Prioritize using MoSCoW
4. Write 2-3 detailed use cases
5. Define 5 non-functional requirements
6. Create acceptance criteria for top 5 user stories

10 Summary

- Requirements are the foundation of software development
- Use multiple techniques to gather requirements
- User stories capture requirements from user perspective
- Use cases detail system interactions
- Prioritize requirements using MoSCoW
- Validate requirements before starting development