

Requirements Engineering Guide

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-14

Table des matières

1 Introduction to Requirements Engineering	2	7.5 Example Story Map	8
2 Requirements Engineering Process Overview	2	8 Domain-Driven Design in Requirements Engineering	8
3 Types of Requirements	3	8.1 Ubiquitous Language	8
3.1 Functional Requirements	3	8.2 Bounded Contexts	8
3.2 Example: Functional Requirement Specification	3	8.3 Example Domain Model	8
3.3 Non-Functional Requirements	3	9 Requirements Traceability	8
4 User Stories	3	9.1 Traceability Framework	8
4.1 User Story Quality Criteria: The INVEST Model	4	9.2 Traceability Tags	9
4.1.a Acceptance Criteria	4	9.3 Requirement Dependencies and Relationships	9
5 Requirements Documentation Methods ..	4	9.4 Traceability Matrix Overview	9
5.1 Scenarios and Acceptance Tests	4	10 Using Requirement Tags	9
5.1.a Example Scenarios:	4	10.1 Priority Tags	10
5.2 Gherkin Specifications and Numbering System	5	10.2 Type Tags	10
5.3 Use Cases	5	10.3 Testing Tags	10
5.4 Data Requirements	5	10.4 Status Tags	10
5.5 External Interface Requirements	6	11 Best Practices for Requirements Engineering	10
5.6 Requirements Specification Documents	6	12 Modern Requirements Practices	10
5.7 System Overview	6	12.1 Requirements in DevOps Environments	10
6 Requirements Gathering Process	7	12.1.a Continuous Requirements ..	10
7 Requirements Visualization Techniques ..	7	12.1.b Requirements as Code	10
7.1 User Journey Maps	7	12.1.c Integration into CI/CD Pipeline	11
7.2 Story Maps	7	12.2 Agile Adaptation	11
7.3 Impact Mapping	7	12.2.a Progressive Refinement	11
7.4 Event Storming	8	12.2.b Living Documentation	11
		12.2.c Just-enough Documentation ..	11
		12.3 AI-Assisted Requirements	11
		13 Requirements Management Tools	11

14	Requirements Templates and Frameworks	11
15	Project Constraints and Assumptions ..	12
15.1	Types of Constraints	12
15.2	Managing Assumptions	12
16	Common Challenges in Requirements Engineering	12
17	Requirements Validation Techniques ...	12
17.1	Review Techniques	12
17.1.a	Peer Reviews	12
17.1.b	Walkthrough Sessions	13
17.1.c	Formal Inspections	13
17.1.d	Stakeholder Reviews	13
17.2	Validation Methods	13
17.2.a	Prototyping	13
17.2.b	Requirements Testing	13
17.2.c	Model Validation	13
17.2.d	Use Case Simulation	14
17.3	Validation Checklist	14
17.3.a	Completeness	14
17.3.b	Consistency	14
17.3.c	Feasibility	14
17.3.d	Testability	14
17.3.e	Traceability	14
17.3.f	Unambiguity	14
17.3.g	Correctness	14
17.4	Validation Documentation	14
18	Requirements Artifacts Relationships ..	15
18.1	Hierarchy and Relationships	15
18.2	Requirements Artifact Comparison	15
18.3	Relationships and Mappings	15
18.4	Key Differences	16
18.5	Example Flow	16
19	Requirements Documentation Types ...	16
19.1	Documentation Structure	17
19.2	Document Dependencies	17
20	Requirements Analysis Techniques ...	17
20.1	Behavior-Driven Development (BDD)	17
20.2	Use Case Modeling	17
21	Requirements Prioritization	18
21.1	Prioritization Methods	18

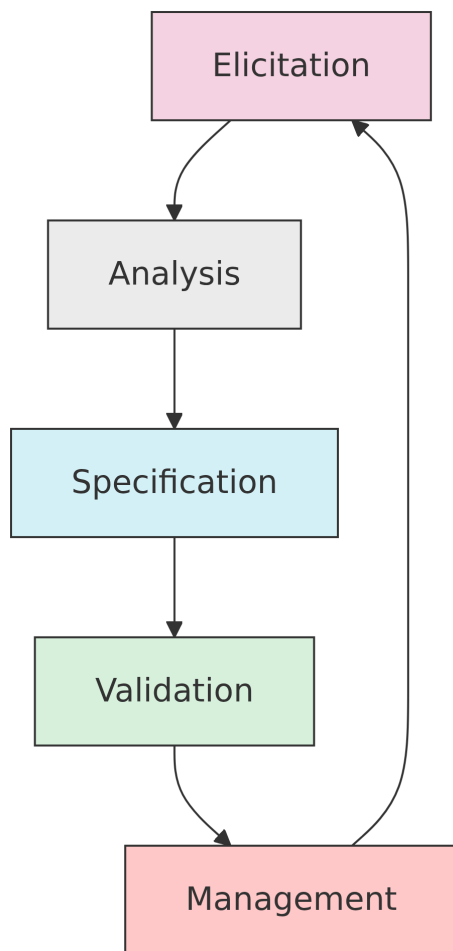
21.2	Release Planning	18
------	------------------------	----

1 Introduction to Requirements Engineering

Requirements Engineering is a systematic approach to eliciting, documenting, validating, and managing requirements throughout the software development lifecycle. This guide provides a structured framework for modern requirements practices, methodologies, and techniques that lead to successful software development outcomes.

2 Requirements Engineering Process Overview

The requirements engineering process consists of iterative activities that transform stakeholder needs into well-defined requirements:



3 Types of Requirements

3.1 Functional Requirements

Functional requirements define specific behaviors, capabilities, and functions that a system must perform to meet user needs.

- Describe what the system should do
- Detail the features and services provided
- Specify behaviors under particular conditions
- Define interactions between users and the system
- Often expressed as « The system shall... »

3.2 Example: Functional Requirement Specification

FR-103: User Authentication

Description: The system must authenticate users before granting access to protected resources.

Details: - The system shall provide a login interface requiring username and password - The system shall validate credentials against the user database - The system shall enforce password complexity requirements - The system shall lock accounts after 5 failed login attempts - The system shall provide a password recovery mechanism

Priority: Must have

Source: Security policy SP-42

Dependent requirements: FR-101, FR-102

Verification method: Security testing

3.3 Non-Functional Requirements

Non-functional requirements define quality attributes, constraints, and characteristics that affect how the system performs its functions.

- Performance: Response times, throughput, resource utilization
- Security: Authentication, authorization, data protection
- Usability: User interface, accessibility, learning curve
- Reliability: Fault tolerance, recoverability, availability
- Compatibility: Platform support, integration capabilities
- Scalability: Growth capacity, load handling
- Maintainability: Ease of updates, modularity, documentation

4 User Stories

User stories are short, simple descriptions of functionality from an end user perspective. They follow this format:

As a [user type], I want [goal] so that [benefit/value].

Example: > As a mobile app user, I want to save my data locally so that I can access it offline.

4.1 User Story Quality Criteria: The INVEST Model

Each user story should meet the INVEST criteria:

- **Independent:** Can be developed and delivered separately
- **Negotiable:** Details can be discussed and refined
- **Valuable:** Delivers specific value to users or stakeholders
- **Estimable:** Team can reasonably estimate development effort
- **Small:** Fits within a single iteration
- **Testable:** Has clear acceptance criteria

4.1.a Acceptance Criteria

Acceptance criteria define when a user story is complete. They should be:

- Written in clear, unambiguous language
- Testable with specific conditions
- Include both happy path and edge cases
- Defined before development begins

Example: > **Given** I am a logged-in user
> **When** I navigate to my profile
> **And** I click the «Edit Profile» button
> **And** I update my profile information
> **And** I click «Save»
> **Then** my updated information should be displayed
> **And** I should see a success message

5 Requirements Documentation Methods

5.1 Scenarios and Acceptance Tests

Scenarios describe specific examples of system behavior using the Given-When-Then format:

```
@test-type @priority @scope
Scenario: E[XX]-F[XX]-S[XX] - [Descriptive Title]
  Given [initial context]
  When [action occurs]
  Then [expected outcome]
```

5.1.a Example Scenarios:

```
@happy-path @smoke @automated
Scenario: E01-F01-S01 - Successful
User Registration
  Given a new user on the registration
  page
  When they complete the registration
  form with valid data
  Then their account should be created
  And they should receive a
  confirmation email
```

```
@edge-case @security
Scenario: E01-F01-S02 - Registration
with Existing Email
  Given a new user on the registration
  page
  And an existing account with email
  "test@example.com"
  When they attempt to register with
  the same email
  Then registration should be
  prevented
  And they should see an error message
```

For detailed scenario templates and examples, see the Scenarios and Acceptance Tests Template.

5.2 Gherkin Specifications and Numbering System

Gherkin specifications should follow a hierarchical numbering format for clear organization:

- EPIC: E[01-99] (e.g., E01)
- Feature: E[01-99]-F[01-99] (e.g., E01-F01)
- User Story: E[01-99]-F[01-99]-US[01-99] (e.g., E01-F01-US01)
- Scenario: E[01-99]-F[01-99]-S[01-99] (e.g., E01-F01-S01)
- Acceptance Test: E[01-99]-F[01-99]-TA[01-99] (e.g., E01-F01-TA01)

Example Structure:

```
## EPIC: E01 - [User Account Management]

### Feature: E01-F01 - [User Registration]

@functional @critical @sprint-1
@user-role-all
**User Story:** E01-F01-US01 - As a new user, I want to create an account so that I can access the application's features.

@happy-path @smoke @automated
Scenario: E01-F01-S01 - Successful Registration
    Given the user is on the registration page
    When the user enters valid registration information
    And submits the registration form
    Then a new account should be created
    And the user should be redirected to the welcome page
```

5.3 Use Cases

Use cases describe interactions between actors and the system to achieve specific goals, typically including:

- Actors involved
- Preconditions
- Main success scenario
- Alternative paths
- Postconditions

5.4 Data Requirements

Data requirements define the structure, relationships, and characteristics of data in the system:

1. Data Structures

- Entity definitions
- Attributes and data types
- Relationships and cardinality
- Data constraints and validation rules

2. Data Quality Requirements

- Accuracy standards
- Completeness requirements
- Consistency rules
- Timeliness requirements

3. Data Lifecycle

- Creation and acquisition
- Storage and retention
- Archival and backup
- Deletion and purging

Example Data Requirement:

```
Entity: Customer
Description: Represents a registered user of the system
Attributes:
  - id: UUID (primary key)
  - email: String (unique, validated)
  - name: String (required)
  - created_at: Timestamp
Relationships:
  - has_many: Orders
```

```
- has_one: ShippingAddress
Constraints:
- email must be unique
- password must meet security policy
Quality:
- 99.99% data accuracy
- < 1s retrieval time
```

For detailed templates and examples, see the Data Requirements Template.

5.5 External Interface Requirements

External interface requirements define how the system interacts with other systems, users, hardware, and software:

1. User Interfaces

- Visual elements and layouts
- Navigation patterns
- Input/output methods
- Accessibility standards

2. System Interfaces

- API specifications
- Data formats
- Communication protocols
- Integration points

Example Interface Requirement:

```
Interface: Payment Gateway API
Type: REST API
Description: Integration with payment processing service
Specifications:
- Protocol: HTTPS
- Authentication: OAuth 2.0
- Data Format: JSON
- Endpoints:
  - POST /api/v1/payments
  - GET /api/v1/transactions
Requirements:
- Max response time: 3 seconds
```

```
- 99.99% availability
- End-to-end encryption
```

For detailed templates and examples, see the External Interface Requirements Template.

5.6 Requirements Specification Documents

Formal documents that detail:

- System overview: Provides a high-level description of the system, its purpose, scope, and objectives
- Functional and non-functional requirements: Details specific behaviors (functional) and quality attributes (non-functional) of the system
- Data requirements: Specifies data structures, relationships, and persistence needs
- External interface requirements: Documents how the system interacts with external components, including APIs, UIs, and hardware interfaces
- Constraints and assumptions: Lists technical, business, and environmental limitations and presumptions that impact the system
- Traceability matrix: Maps relationships between requirements, design components, and test cases to ensure complete coverage

5.7 System Overview

The system overview provides a high-level understanding of the system's purpose, scope, and key components:

1. System Purpose and Scope

- Primary goals and objectives
- Target users and stakeholders
- Key value propositions
- Business objectives

2. System Context

```
graph TD
  A[System] --> B[External System 1]
  A --> C[External System 2]
  A --> D[External System 3]
  E[User Type 1] --> A
  F[User Type 2] --> A
```

3. **Technology Stack** | Layer | Technologies | Rationale | |---|-----|----| | Frontend | [Technologies] | [Justification] | | Backend | [Technologies] | [Justification] | | Database | [Technologies] | [Justification] | | Infrastructure | [Technologies] | [Justification] |

For detailed templates and examples, see the System Overview Template.

6 Requirements Gathering Process

An effective requirements gathering process typically includes:

1. **Stakeholder Identification:** Determine who has input on or is affected by the system.
2. **Elicitation Techniques:**
 - Interviews with stakeholders
 - Workshops and focus groups
 - Surveys and questionnaires
 - Observation of current processes
 - Analysis of existing documentation
 - Prototyping and user feedback
3. **Analysis and Refinement:**
 - Remove ambiguities and resolve conflicts
 - Identify gaps and inconsistencies
 - Elaborate high-level requirements into detailed specifications
 - Validate requirements with stakeholders
4. **Documentation:**

- Create formal requirements documentation using appropriate templates
- Establish traceability between requirements
- Manage requirements versioning

5. Prioritization:

- Apply the MoSCoW method (Must have, Should have, Could have, Won't have)
- Consider business value, risk, dependencies, effort, and time constraints
- Create a prioritized backlog for implementation

6. Validation and Verification:

- Review requirements with stakeholders
- Ensure requirements are testable
- Check for completeness and consistency
- Validate alignment with business goals

7 Requirements Visualization Techniques

Visual representations often communicate requirements more effectively than text alone:

7.1 User Journey Maps

Visualize the end-to-end user experience through a service or product, highlighting touchpoints, emotions, and pain points.

7.2 Story Maps

Organize user stories into a narrative flow that shows the user journey horizontally and details vertically.

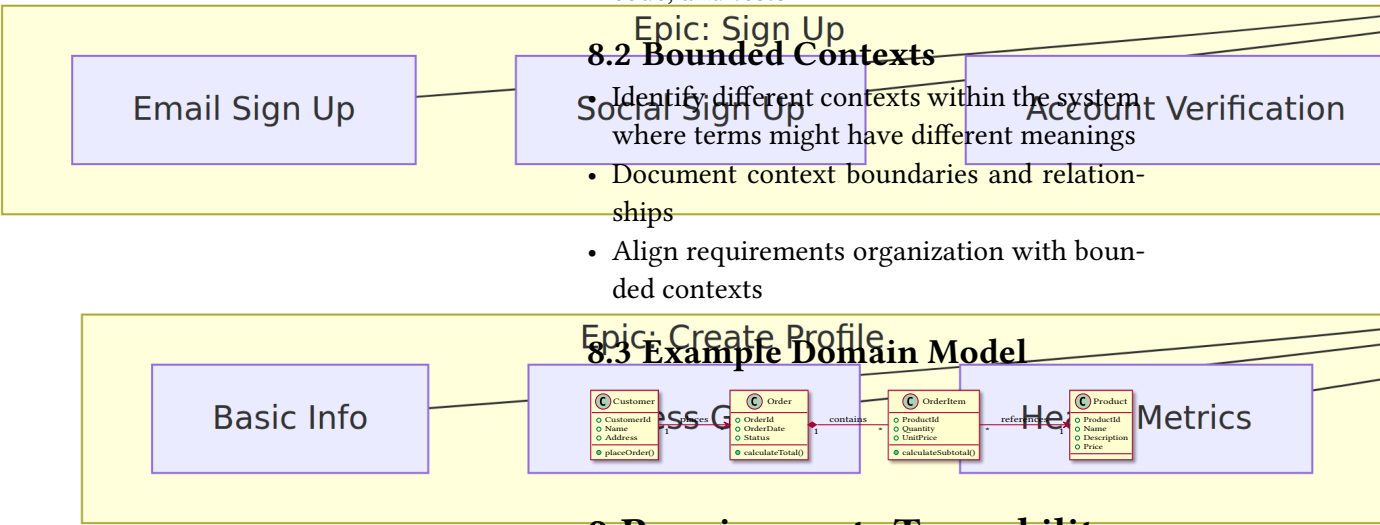
7.3 Impact Mapping

Connect business goals to user needs, deliverables, and development activities in a hierarchical diagram.

7.4 Event Storming

Collaborative modeling technique using sticky notes to explore domain events, commands, aggregates, and policies.

7.5 Example Story Map



8 Domain-Driven Design in Requirements Engineering

Domain-Driven Design (DDD) provides a powerful approach to requirements engineering by focusing on the core domain and domain logic.

8.1 Ubiquitous Language

- Create a shared language between developers and domain experts
- Document domain terminology in a glossary
- Use consistent terminology in requirements, code, and tests

8.2 Bounded Contexts

- Identify different contexts within the system where terms might have different meanings
- Document context boundaries and relationships
 - Align requirements organization with bounded contexts

8.3 Example Domain Model

9 Requirements Traceability

9.1 Traceability Framework

A comprehensive traceability framework should include:

Re- qui- re- ment ID	Source User Story	De- sign Docu- ment	Im- ple- men- ta- tion	Test Case tus
E01- F01- S01	Client Meeting US01 #2	DB Schema v1.2	User Control	Full Met F01- ri- TA01 fied

9.2 Traceability Tags

Use these special tags to establish traceability links:

- `@source`: Links to requirement source (e.g., `@source:PRD-v1.2`)
- `@implements`: Links to design document (e.g., `@implements:design-doc-login`)
- `@code`: Links to implementation (e.g., `@code:com.example.UserController`)
- `@tests`: Links to test implementation (e.g., `@tests:UserRegistrationTest`)

9.3 Requirement Dependencies and Relationships

Document relationships between requirements using these conventions:

Relationship	Syntax	Description
Depends on	<code>@depends-on:E01-F02</code>	Requirement depends on another being implemented first
Extends	<code>@extends:F02</code>	Requirement extends functionality of another
Conflicts with	<code>@conflicts-with:E01-F02</code>	Potential conflict needing resolution
Replaces	<code>@replaces:F02</code>	Requirement supersedes an earlier one
Related to	<code>@related-to:E01-F02</code>	General relationship with another requirement

9.4 Traceability Matrix Overview

A comprehensive traceability matrix maps relationships between:

1. **Business Requirements to System Requirements**
 - Links high-level business needs to specific system features
 - Ensures complete coverage of business objectives
 - Identifies gaps in implementation
2. **Requirements to Design Components**
 - Maps requirements to architectural components
 - Validates technical implementation approach
 - Tracks design coverage
3. **Requirements to Test Cases**
 - Links requirements to verification methods
 - Ensures complete test coverage
 - Validates requirement testability

Example Matrix:

Re-quire-ment ID	Busi-ness Req	User Sto-ry	De-sign Doc	Test Case	Sta-tus
FR-103	BR-001	US-001	DE-SIGN-001	TC-001	Ver-ified
NFR-201	BR-002	US-002	DE-SIGN-002	TC-002	In Pro-gress

For detailed templates and examples, see the Traceability Matrix Template.

10 Using Requirement Tags

Tags serve multiple purposes in requirements management:

10.1 Priority Tags

- **@critical** - Must be implemented for core functionality
- **@high** - Important feature needed soon
- **@medium** - Standard priority feature
- **@low** - Nice-to-have feature

10.2 Type Tags

- **@functional** - Core behavior functionality
- **@non-functional** - Quality attributes
- **@ui** - User interface related
- **@api** - API functionality
- **@data** - Data handling
- **@integration** - External system integration

10.3 Testing Tags

- **@smoke** - Basic functionality verification
- **@regression** - Regression detection
- **@automated** - Automated test scenarios
- **@manual** - Manual test scenarios
- **@performance** - Performance testing
- **@security** - Security testing
- **@accessibility** - Accessibility testing
- **@usability** - Usability testing

10.4 Status Tags

- **@wip** - Work in progress
- **@blocked** - Implementation blocked
- **@implemented** - Feature implemented
- **@verified** - Feature tested and verified

11 Best Practices for Requirements Engineering

- **Involve the right stakeholders:** Ensure all relevant perspectives are considered.
- **Use clear, unambiguous language:** Avoid vague terms like « user-friendly, » « fast, » or « efficient » without measurable criteria.
- **Be specific and measurable:** Define quantifiable criteria for non-functional requirements.

- **Maintain traceability:** Link requirements to business goals, design elements, code implementation, and test cases.
- **Iteratively refine requirements:** Requirements evolve; plan for ongoing refinement.
- **Document assumptions and constraints:** Make implicit limitations explicit.
- **Use visual aids:** Diagrams, mockups, and prototypes can clarify complex requirements.
- **Establish a glossary:** Define domain-specific terms to ensure shared understanding.
- **Consider edge cases and exceptions:** Document how the system should handle unusual situations.
- **Validate with acceptance criteria:** Each requirement should have clear criteria for determining when it's met.

12 Modern Requirements Practices

12.1 Requirements in DevOps Environments

Modern DevOps practices change how requirements are managed:

12.1.a Continuous Requirements

- Requirements evolve alongside continuous delivery
- Feedback loops inform requirement adjustments
- Feature flags enable progressive delivery of requirements
- A/B testing validates requirement assumptions

12.1.b Requirements as Code

- Version control requirements alongside code
- Automated validation of requirements

- Requirements expressed as executable specifications
- Infrastructure requirements defined as code

12.1.c Integration into CI/CD Pipeline

- Automated requirements validation
- Requirements compliance verification
- Traceability enforcement
- Release notes generation from requirements changes

12.2 Agile Adaptation

12.2.a Progressive Refinement

Progressive refinement is a core agile practice that involves:

- Starting with high-level epics and gradually breaking them down
- Regular refinement sessions with stakeholders
- Continuous evolution of requirements based on feedback
- Using story splitting techniques for complex requirements
- Maintaining a healthy backlog refinement cadence

12.2.b Living Documentation

Living documentation keeps requirements aligned with the evolving system:

- Automated testing as documentation (Behavior-Driven Development)
- Wiki-based requirements that evolve with the product
- Version-controlled requirements alongside code
- Automated generation of documentation from code and tests
- Regular review and updates based on implementation feedback

12.2.c Just-enough Documentation

- Balance between documentation and agility

- Focus on essential information capture
- Documentation that serves a clear purpose

12.3 AI-Assisted Requirements

- Natural language processing for requirement analysis
- Automated consistency checking
- Smart requirement classification

13 Requirements Management Tools

Various tools can help with requirements gathering and management:

- Specialized requirements management tools (e.g., Jira, Azure DevOps, ReqSuite)
- Documentation tools with requirements templates (e.g., Confluence, SharePoint)
- Modeling tools for visual requirements (e.g., Lucidchart, Draw.io)
- Collaborative workspaces (e.g., Miro, Mural) for requirements workshops
- Version control systems for requirements documentation

14 Requirements Templates and Frameworks

For detailed templates and frameworks to document requirements effectively, see our standardized templates in the Requirements Templates Guide:

- System Requirements Specification
- System Overview
- Requirements Glossary
- Functional Requirements
- Non-functional Requirements
- User Stories
- Use Cases
- Data Requirements
- Interface Requirements
- Constraints and Assumptions
- Requirements Traceability

15 Project Constraints and Assumptions

Understanding and documenting constraints and assumptions is crucial for successful requirements engineering:

15.1 Types of Constraints

1. Business Constraints

- Regulatory requirements
- Budget limitations
- Timeline restrictions
- Resource availability

2. Technical Constraints

- Hardware limitations
- Software dependencies
- Integration requirements
- Performance boundaries

3. Environmental Constraints

- User environment
- System environment
- Infrastructure requirements
- Deployment restrictions

15.2 Managing Assumptions

1. Documentation

- Clearly state each assumption
- Provide rationale
- Identify risks
- Plan validation approach

2. Validation Process

- Regular review of assumptions
- Update as new information emerges
- Track impact on requirements
- Maintain traceability

For detailed templates and guidance on documenting constraints and assumptions, see the Constraints and Assumptions Template.

16 Common Challenges in Requirements Engineering

- **Scope creep:** Continuously expanding requirements
- **Unstated assumptions:** Implicit requirements that aren't documented
- **Conflicting requirements:** Contradictory needs from different stakeholders
- **Analysis paralysis:** Excessive detail slowing down the process
- **Ambiguity:** Unclear or vague specifications
- **Over-specification:** Dictating implementation rather than requirements
- **Changing requirements:** Managing evolving needs during development

Addressing these challenges requires strong communication, stakeholder engagement, appropriate methodology selection, and effective requirements management tools and processes.

17 Requirements Validation Techniques

Requirements validation ensures that the documented requirements accurately represent stakeholder needs and can be implemented into a working system that meets those needs.

17.1 Review Techniques

17.1.a Peer Reviews

- **Description:** Systematic examination of requirements by peers and colleagues
- **Benefits:** Identifies inconsistencies and ambiguities early
- **Process:**
 1. Distribute requirements to reviewers
 2. Allow time for individual review
 3. Collect and consolidate feedback
 4. Address issues identified

17.1.b Walkthrough Sessions

- **Description:** Presenter guides stakeholders through requirements step by step
- **Benefits:** Creates shared understanding and surfaces questions
- **Process:**
 1. Present requirements in logical order
 2. Encourage questions and discussion
 3. Document concerns and suggested improvements
 4. Follow up with revisions

17.1.c Formal Inspections

- **Description:** Structured review process with defined roles and procedures
- **Benefits:** High defect detection rate, provides metrics on requirements quality
- **Roles:**
 - Moderator: Manages the process
 - Reader: Presents the material
 - Recorder: Documents issues
 - Inspector: Reviews for specific issues
- **Process:**
 1. Planning and preparation
 2. Inspection meeting
 3. Rework
 4. Follow-up verification

17.1.d Stakeholder Reviews

- **Description:** Validation sessions with business stakeholders
- **Benefits:** Ensures requirements meet business needs, builds buy-in
- **Best Practices:**
 - Prepare executive summaries for high-level stakeholders
 - Use visual aids to help explain technical concepts
 - Document sign-off and approval

17.2 Validation Methods

17.2.a Prototyping

- **Types:**

- Low-fidelity: Paper sketches, wireframes
- Medium-fidelity: Interactive mockups
- High-fidelity: Working prototypes

- **Benefits:**

- Makes abstract requirements tangible
- Allows early user feedback
- Reveals unstated requirements

- **Example Process:**

1. Create prototype based on requirements
2. Conduct user testing sessions
3. Collect feedback
4. Refine requirements based on insights

17.2.b Requirements Testing

- **Description:** Creating test cases directly from requirements

- **Techniques:**

- Requirements-based test design
- Boundary value analysis
- Equivalence partitioning

- **Example:**

```
Requirement:  The system shall
accept passwords between 8-20
characters
```

```
Test Cases:
```

- 7 characters (invalid)
- 8 characters (valid)
- 20 characters (valid)
- 21 characters (invalid)

17.2.c Model Validation

- **Description:** Validating requirements through models and simulations

- **Modeling Techniques:**

- UML diagrams
- Business process models
- State machines
- Data models

- **Validation Approach:**

1. Create models representing requirements

2. Walk through scenarios using models
3. Verify logic and completeness
4. Update requirements based on model insights

17.2.d Use Case Simulation

- **Description:** Mentally executing use cases step by step
- **Benefits:**
 - Validates flow and logic
 - Identifies missing steps and conditions
 - Tests alternative paths
- **Process:**
 1. Select a use case
 2. Define test scenarios
 3. Execute each step mentally or with simple tools
 4. Document issues and gaps found

17.3 Validation Checklist

A comprehensive requirements validation should verify the following qualities:

17.3.a Completeness

- All required functionality is specified
- All interfaces are defined
- All constraints are documented
- No «TBD» (To Be Determined) items remain without explanation

17.3.b Consistency

- No contradictions between requirements
- Consistent terminology throughout
- Consistent level of detail
- Alignment with higher-level requirements

17.3.c Feasibility

- Technical feasibility assessment
- Schedule feasibility evaluation
- Budget feasibility check
- Operational feasibility confirmation

17.3.d Testability

- Each requirement can be verified objectively

- Clear acceptance criteria exist
- Quantifiable metrics where appropriate
- Test cases can be derived

17.3.e Traceability

- Each requirement is uniquely identified
- Links to source (business need, stakeholder, etc.)
- Forward traceability to design and tests
- Backward traceability to higher-level requirements

17.3.f Unambiguity

- Clear, precise language
- Measurable criteria where applicable
- No subjective terms without definition
- Single interpretation possible

17.3.g Correctness

- Reflects stakeholder needs accurately
- Aligns with business objectives
- Conforms to standards and regulations
- Technically sound

17.4 Validation Documentation

Document the results of validation activities to ensure traceability and accountability:

1. Validation Report

- Summary of validation activities
- Issues identified
- Resolution actions
- Validation status of each requirement

2. Review Record

- Date and participants
- Requirements reviewed
- Comments and issues raised
- Decisions made

3. Formal Sign-off

- Stakeholder acknowledgment
- Approval signatures
- Conditions of approval
- Date of validation

18 Requirements Artifacts Relationships

18.1 Hierarchy and Relationships

```
graph TD
    E[Epic] --> F[Feature]
    F --> US[User Story]
    US --> FR[Functional Requirement]
    US --> NFR[Non-Functional Requirement]
    FR --> S[Scenario]
    NFR --> S
    US --> UC[Use Case]
    UC --> S
    S --> AT[Acceptance Test]
```

18.2 Requirements Artifact Comparison

As-pect	User Story	Use Case	Functional Req	Non-Functional Req	Scenario
Focus	User value	User interaction	System behavior	System qualities	Specific example
Format	As a/I want/ So that	Steps & flow	System shall	System shall be	Given/ When/ Then
Scope	Single feature slice	Complete interaction	Specific function	Quality attribute	Single interaction path
Level	Business	User interaction	Technical	Technical	Implementation

As-pect	User Story	Use Case	Functional Req	Non-Functional Req	Scenario
Purposing	Planning	Design	Specification	Constraints	Validation

18.3 Relationships and Mappings

1. Epic to Features

- Epic: Large initiative (e.g., « User Account Management »)
- Features: Major components (e.g., « Registration », « Authentication »)

2. Features to User Stories

```
Feature: E01-F01 - User Registration

User Story: E01-F01-US01 - As a new user...
User Story: E01-F01-US02 - As a registered user...
```

3. User Stories to Requirements

```
User Story: E01-F01-US01
@implements-fr:FR-103 // Authentication requirement
@implements-nfr:NFR-201 // Response time requirement
```

4. Requirements to Scenarios

```
FR-103: User Authentication

Scenario: E01-F01-S01 - Successful Login
Scenario: E01-F01-S02 - Failed Login
```

5. Use Cases to Scenarios

```
Use Case: UC-01 - User Login Process
```

```
Main Scenario: E01-F01-S01 - Successful Login
```

```
Alternative: E01-F01-S02 - Failed Login
```

```
Exception: E01-F01-S03 - Account Locked
```

18.4 Key Differences

1. User Stories vs Use Cases

- User Stories: Brief, focused on value
- Use Cases: Detailed, focused on interaction steps

2. Functional vs Non-Functional Requirements

- Functional: What the system does
- Non-Functional: How well the system does it

3. Scenarios vs Use Cases

- Scenarios: Specific instances/examples
- Use Cases: Complete interaction flows

4. Requirements vs Tests

- Requirements: Specify what's needed
- Tests: Verify implementation meets requirements

18.5 Example Flow

```
# Epic  
E01 - User Account Management
```

```
# Feature  
E01-F01 - User Registration
```

```
# User Story  
E01-F01-US01 - As a new user, I want to create an account...
```

```
# Functional Requirement  
FR-103 - The system shall authenticate users...
```

```
# Non-Functional Requirement  
NFR-201 - Registration process shall complete within 2 seconds...
```

```
# Use Case  
UC-01 - User Registration Process  
1. User navigates to registration  
2. System displays form  
3. User submits information
```

```
...
```

```
# Scenario  
E01-F01-S01 - Successful Registration  
Given a new user on the registration page  
When they submit valid information  
Then their account is created
```

```
# Acceptance Test  
E01-F01-TA01 - Verify Registration Success  
Given test user data  
"test@example.com"  
When registration form is submitted  
Then database contains new user  
And welcome email is sent
```

19 Requirements Documentation Types

Requirements documentation follows a hierarchical structure:

1. System Requirements Specification (SRS)

- Complete system documentation
- High-level overview
- All requirement types

2. User Stories and Use Cases

- Business value focused
- User interaction flows
- Acceptance criteria

3. Technical Requirements

- Functional specifications
 - Non-functional attributes
 - Interface definitions
4. **Supporting Documentation**
- Data requirements
 - Constraints and assumptions
 - Traceability information

19.1 Documentation Structure

```
graph TD
    SRS[System Requirements Specification] --> US[User Stories]
    SRS --> UC[Use Cases]
    SRS --> FR[Functional Requirements]
    SRS --> NFR[Non-Functional Requirements]
    SRS --> DR[Data Requirements]
    SRS --> IR[Interface Requirements]
    US --> AC[Acceptance Criteria]
    UC --> SC[Scenarios]
    FR --> TS[Test Specifications]
    NFR --> VM[Validation Methods]
```

19.2 Document Dependencies

Document Type	Depends On	Influences
SRS	Business Requirements	All other docs
User Stories	SRS	Functional Reqs
Use Cases	User Stories	Scenarios
Functional Reqs	User Stories	Test Specs
Non-Functional Reqs	SRS	Validation

20 Requirements Analysis Techniques

20.1 Behavior-Driven Development (BDD)

Principle: Define expected system behaviors through scenarios understandable by all stakeholders.

Implementation: - Use of Gherkin format (Given-When-Then) - Example Mapping sessions to develop scenarios - Automated tests directly linked to specifications

Recommended Tools: - Cucumber, Specflow, JBehave - FitNesse for acceptance testing

References: - «BDD in Action» by John Ferguson Smart - «Specification by Example» by Gojko Adzic

20.2 Use Case Modeling

Principle: Document interactions between actors and the system to achieve specific goals.

Recommended Structure: - Primary and secondary actors - Pre and post conditions - Main flow and alternatives - Extensions and exceptions

Documentation Format:

```
Use Case: [Use Case Name]
Primary Actor: [Actor]
Preconditions: [Required Conditions]
Main Flow:
1. [Step 1]
2. [Step 2]
...
Alternative Flows:
- [Condition]: [Action Sequence]
Postconditions: [System State After Execution]
```

References: - «Writing Effective Use Cases» by Alistair Cockburn - «UML Distilled» by Martin Fowler

21 Requirements Prioritization

21.1 Prioritization Methods

Recommended Techniques:

1. **MoSCoW:**
 - Must have: Essential features
 - Should have: Important features
 - Could have: Desirable features
 - Won't have: Postponed features
2. **Weighted Scoring:**
 - Business value (30-50%)
 - Technical complexity (20-30%)
 - Risk (15-25%)
 - Cost (10-20%)
3. **User Story Mapping:**
 - Horizontal organization by user activities
 - Vertical organization by priority
 - Identification of coherent releases

References: - «User Story Mapping» by Jeff Patton - «Software Requirements» by Karl Wiegers and Joy Beatty

21.2 Release Planning

Recommended Structure: - MVP (Minimum Viable Product): Core features only - Incremental releases: 4-6 weeks per release - Priority reassessment after each user feedback