Course: Software Requirements Engineering

Week 10: Writing Requirements

Lecturer: Yimer Amedie (MSc.)

Addis Ababa Science and Technology University

May, 2026

Contents

- Introduction
- Principles of Writing Clear & Effective Requirements
- Techniques for Ensuring Consistency & Verifiability
- Guidelines for Reducing Ambiguity in Requirements
- Writing FRs and NFRs
- Acceptance Criteria and Requirement Quality Review Techniques

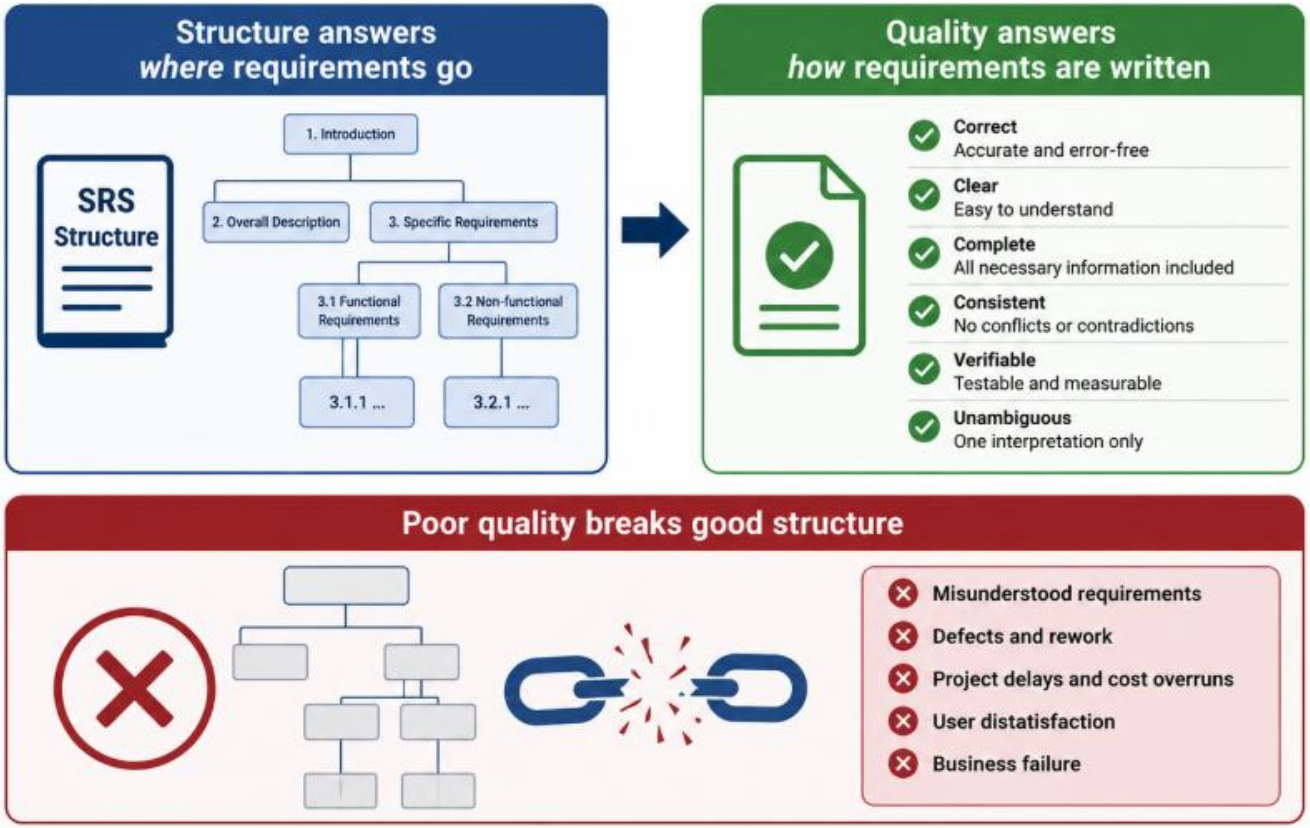
Figure 1. *The software requirements*
Note. Image generated using Sora by OpenAI (2026).

Learning Outcomes

After completing this lesson, you will be able to:

- Write clear, concise, and unambiguous software requirements
- Apply consistency and verifiability principles in requirement writing
- Distinguish between well-written and poorly written requirements
- Develop measurable acceptance criteria for software requirements
- Evaluate and improve requirement quality using standard writing guidelines

Introduction



Writing is a critical engineering skill

Well writing requirements turn good structure into successful system

Writing Requirements

- ✓ Writing requirements is the process of transforming analyzed needs into clear, structured, and precise statements.
  - ➢ Transform ideas into formal statements
  - ➢ Focus on clarity and precision
  - ➢ Avoid misinterpretation
  - ➢ Support development and testing
  - ➢ Critical for project success

Goal of Writing Requirements

✓ Two important goals of writing requirements are that (Beatty, 2013):

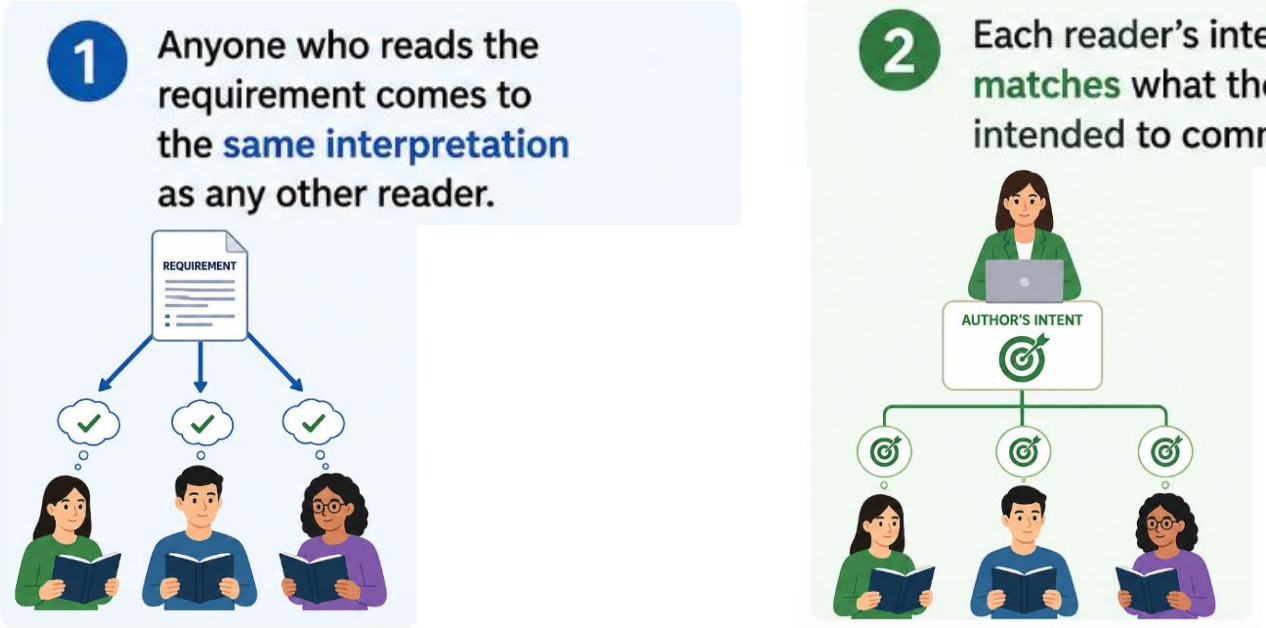
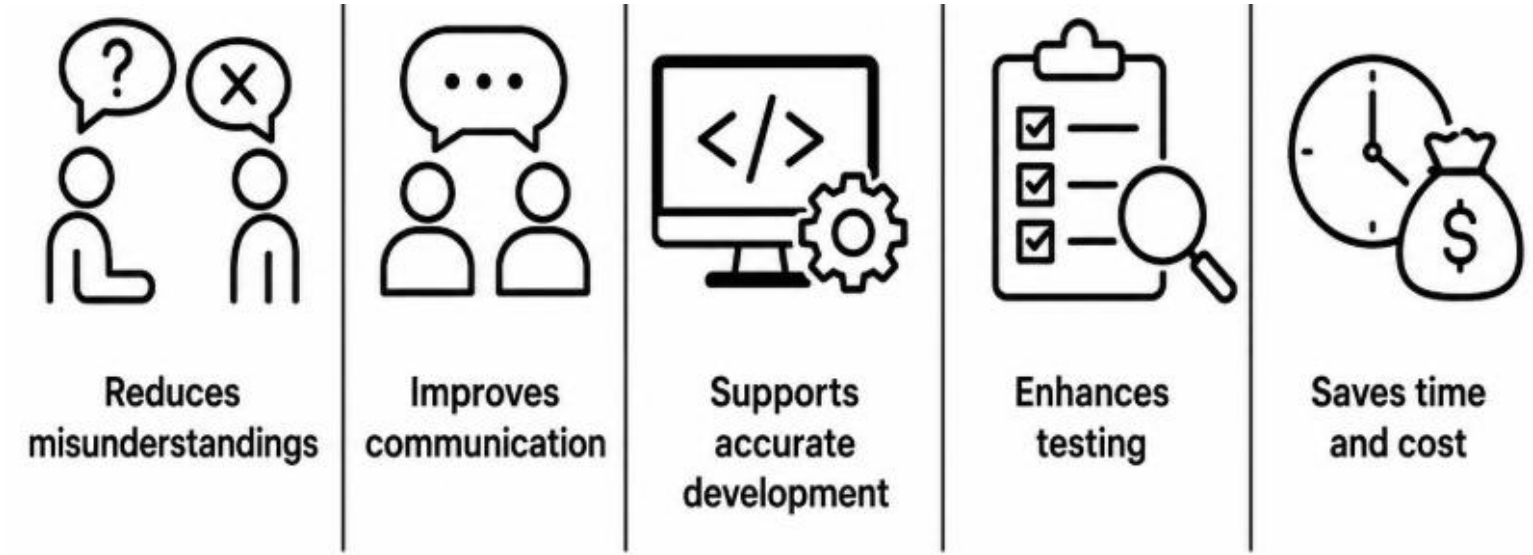


Figure 2. *Goal of writing requirements*
Note. Image generated using ChatGPT by OpenAI (2026).

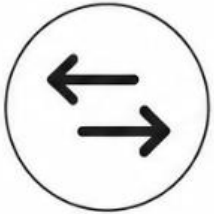
Importance of Clear Requirements



Problems in Poorly Written Requirements

Ambiguous wording

Vague or unclear language leads to multiple interpretations.



Contradictory statements

Conflicting information creates confusion and errors.

Unverifiable requirements

Cannot be tested or measured, making validation impossible.

Mixed design and requirements

Solution or design details are mixed with what is needed.

Missing acceptance criteria

No clear criteria to determine when requirements are satisfied.

Clear requirements prevent rework, reduce risks, and lead to successful outcomes.

Key Characteristics

Well-written requirements share several important characteristics.

Clarity: What Does It Mean?

Single, precise meaning

Simple and direct language

Define terms clearly

No unnecessary complexity

Use consistent terminology

Write short sentences

Understandable by all stakeholders

A clear requirement answers three questions:

1. What the system shall do?
2. Under what conditions?
3. For whom?

Writing Clear Requirement Statement

Use “The system shall...”

One requirement per statement

Use active voice

Avoid compound sentences

Examples: Clear vs. Unclear Requirements

Unclear example:

"The system shall respond quickly to user requests."
This statement is unclear because "quickly"

Problems identified:

- Different stakeholders may have different expectations

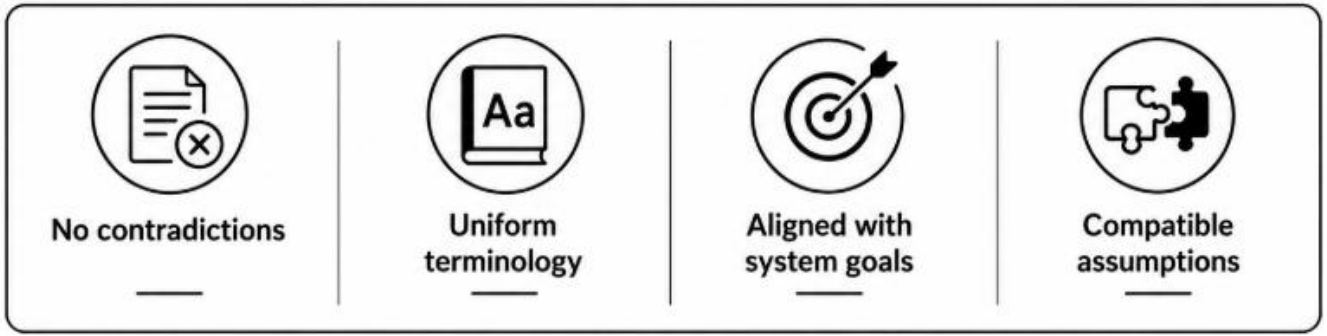
Improved version:

"The system shall display search results within 2 seconds for 95% of requests under normal operating conditions."

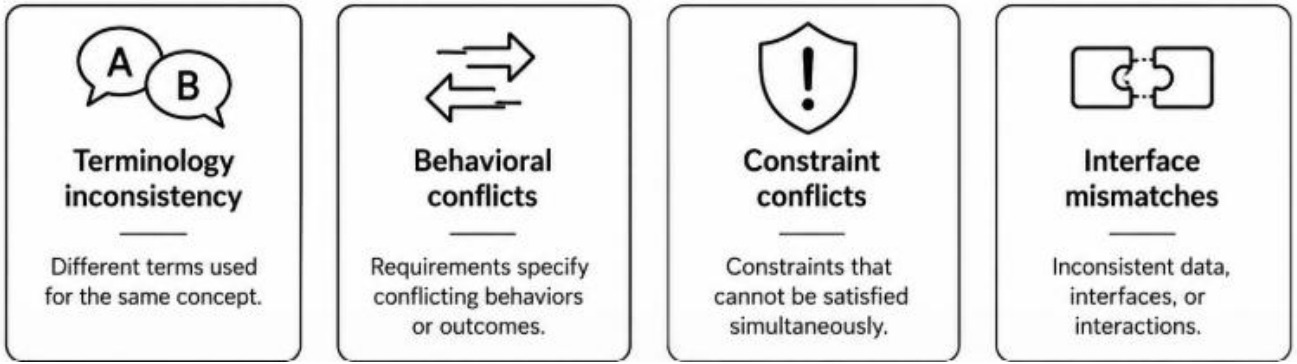
Key lesson:

Clarity often requires specificity

Consistency in Software Requirements



Types of Inconsistencies



Techniques to Ensure Consistency

Several practical techniques help ensure consistency.

Use a glossary

Maintain a glossary of domain terms and definitions to ensure everyone uses the same terminology consistently.

Apply requirement templates

Use standard requirement templates to ensure all requirements are captured in a uniform structure and format.

Conduct peer reviews

Review requirements with peers to catch ambiguities, gaps, and inconsistencies early and ensure shared understanding.

Trace requirements to goals

Link each requirement to business goals and objectives to ensure they remain relevant, complete, and consistent.

Verifiability of Requirements

Testable statements

Measurable criteria

Objective evaluation

Clear expected outcomes

Supports validation

Verifiability means that a requirement can be checked to determine whether it has been satisfied.

Example: Verifiable vs. Non-Verifiable Requirements

Non-verifiable:

"The system shall be user-friendly."
This fails because "user-friendly"

Reason it fails:

- Its subjective

Verifiable rewrite:

"New users shall be able to complete the registration process within 3 minutes without external assistance."

Key lesson:

verifiability requires concrete evidence,
not opinions or impressions

Relationship with Testing

- Requirements drive test cases
- Each requirement needs a test
- Gaps reveal weak requirements
- Early test thinking improves quality

Avoiding Ambiguity

A situations where a requirement can be interpreted in more than one way.

Occurs when a requirement can be interpreted in multiple ways.

Source:

- Vague adjectives
- Relative terms
- Undefined references
- Compound statements

Writing FRs and NFRs

FRs

- ✓ Describe system behavior
- ✓ Specify inputs and outputs
- ✓ Define processing logic
- ✓ Use clear conditions
- ✓ Avoid ambiguity

NFRs

- ✓ Define quality attributes
- ✓ Include measurable criteria
- ✓ Cover performance and security
- ✓ Specify constraints, Ensure testability

2 **Non-Functional Requirements:** Describe system qualities

Example: "The system shall respond within 2 seconds."

1 **Functional Requirements:** Describe system behavior

Example: "The system shall allow users to transfer funds between accounts."

Writing Requirements Specification Guidelines

Writing Guideline … (1/4)

Requirement Identification

Assigning unique identifiers to each requirement is essential for effective management and traceability.

- ✓ Supports tracking
- ✓ Enables traceability
- ✓ Simplifies management
- ✓ Avoids confusion

Writing Guideline … (2/4)

Writing Testable Requirements

- ✓ Include measurable metrics
- ✓ Define clear outcomes
- ✓ Avoid subjective terms
- ✓ Support test case creation
- ✓ Ensure validation

Writing Constraints Clearly

- ✓ Define system limitations
- ✓ Specify boundaries
- ✓ Include technical constraints
- ✓ Consider regulatory factors
- ✓ Ensure clarity

Writing Guideline … (3/4)

Requirement Prioritization in Writing

- ✓ Identify critical requirements
- ✓ Rank importance
- ✓ Focus on value
- ✓ Support decision-making
- ✓ Align with goals

Guidelines for Readability

- ✓ Use simple sentences
- ✓ Avoid long paragraphs
- ✓ Use bullet points
- ✓ Maintain logical flow
- ✓ Use headings

Writing Guideline … (4/4)

Use of Active Voice

- ✓ Clear responsibility
- ✓ Direct statements
- ✓ Easier understanding
- ✓ Improves clarity
- ✓ Preferred style

Handling Edge Cases

- ✓ Consider unusual scenarios
- ✓ Define system behavior
- ✓ Improve robustness
- ✓ Prevent failures
- ✓ Enhance completeness

Documentation Style Guidelines

Consistent formatting
Use uniform fonts, headings, spacing, and alignment throughout.

Standard terminology
Use approved terms and definitions consistently across all documents.

Clear structure
Organize content logically with clear headings, sections, and numbering.

Professional tone
Use neutral, respectful, and concise language. Avoid slang and bias.

Follow standards
Comply with relevant organizational, industry, and documentation standards.

Use of Standard Requirement Format

A structured way to document requirements so they are clear, complete, testable, and consistent across projects.

The <system/component> shall <action/function> <condition/constraint>.

The [system] shall [do something] [under conditions].

Example:-

- ✓ **The inventory system shall *update stock levels* immediately after a completed sale.**

Requirement Statement Structure

Requirement Statement Structure

1. Actor + action + object
2. Conditions and constraints
3. Expected outcome
4. Clear boundaries
5. Unique identifier

Requirement Writing Format

A good requirement often follows:

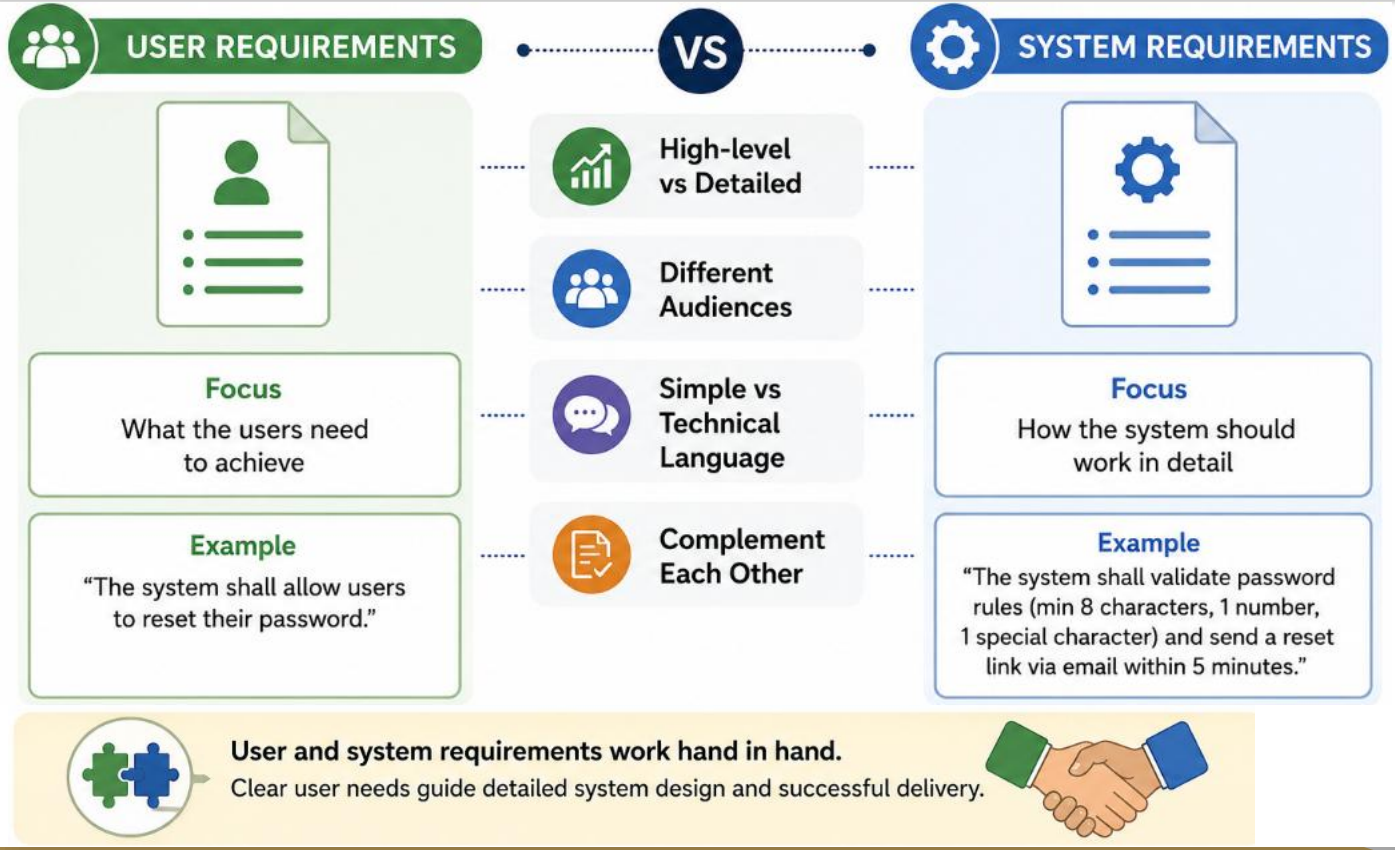
"The system shall [action] [object] [condition]."

Examples:

Poor: "System should be fast."

Good: "The system shall process transactions within 2 seconds under normal load."

User Vs System Requirements Writing



FRs and NFRs Structure

FRs

- ✓ Feature-based organization
- ✓ Input-process-output format
- ✓ Use cases or scenarios
- ✓ Condition-response format
- ✓ Unique identifiers

NFRs

- ✓ Performance requirements
- ✓ Security requirements
- ✓ Usability requirements
- ✓ Reliability requirements
- ✓ Scalability requirements

Practical Checklist for Writing Requirements

Is the requirement

Clear? Easy to understand and unambiguous.

Specific? Well-defined and detailed.

Testable? Can be verified through testing.

Measurable? Quantifiable or has clear acceptance criteria.

Feasible? Achievable with available resources and technology.

Necessary? Adds real value and is worth implementing.

Consistent? Aligned with other requirements and business rules.

Complete? Includes all necessary information.

Traceable? Can be traced to business needs and design elements.

Writing Guidelines – Final Remark!

Best Practices

Clarity: Avoid vague words like *fast, user-friendly, efficient*

Completeness: Include all necessary details

Consistency: No conflicting requirements

Verifiability: Must be testable

Traceability: Link requirements to source and design

Common Mistakes & Writing Tips

Common Mistakes

- Ambiguous wording
- Mixing requirements with design
- Overly complex sentences
- Missing edge cases

Writing Tips

- Use simple and precise language
- Use numbered requirements
- Use “shall” instead of “should”
- Avoid subjective terms
- Keep requirements atomic (one idea per statement)

Acceptance Criteria

Defines success conditions

Clearly states what needs to be achieved for the requirement.

Linked to requirements

Directly connected to the specific requirement it validates.

Specifies expected results

Describes the expected outcome or behavior in measurable terms.

Supports testing

Provides a basis for creating test cases and test scenarios.

Usually created during:

- ✓ Requirements analysis
- ✓ Backlog refinement
- ✓ User story definition
- ✓ Stakeholder discussions

Acceptance Criteria … cont'd

Ensures clarity	Reduces ambiguity and ensures shared understanding.
Improves validation	Helps verify that the solution meets stakeholder needs.
Clarifies "done"	Defines when the requirement can be considered complete.

Answer:

How do we know this requirement has been correctly implemented?

They are closely connected to:

- ✓ functional requirements
- ✓ validation and verification
- ✓ software testing
- ✓ quality assurance

Acceptance Criteria … cont’d

Characteristics of Good Acceptance Criteria

- ✓ Clear and specific
- ✓ Testable
- ✓ Measurable
- ✓ Relevant to requirement

Why Acceptance Criteria Are Important?

- ➢ Reduces disputes
- ➢ Guides testing
- ➢ Aligns expectations
- ➢ Improves quality control

Acceptance Criteria – Example

"As a user, I want to reset my password so that I can regain access to my account."

These criteria define what "done" means.

Acceptance Criteria

- User can **request a password reset** using their email address.
- A reset link is sent to the registered email **within 1 minute.**
- Reset link expires after **24 hours.**
- New password must contain:
  - at least 8 characters
  - one uppercase letter
  - one number
- User can **log in successfully** with the new password.

Acceptance Criteria – Common Format

1 Bullet List

Simple and direct.

Example:

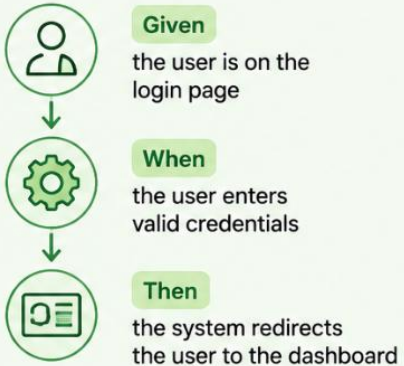
- System shall allow login with valid credentials.
- Error message appears for invalid password.

2 Given–When–Then Format (BDD Style)

Popular in Agile Software Development and behavior-driven development.

Example:

Given	the user is on the login page
When	the user enters valid credentials
Then	the system redirects the user to the dashboard



This format improves clarity and testability.

Acceptance Criteria Vs Test Cases

Acceptance Criteria		Test Cases
Define **conditions** (what should be true)	↔	Define **procedures** (how to verify)
Complementary roles	🤝	Complementary roles
Business / user-focused (high-level)	📊	Technical / tester-focused (detailed)
Higher abstraction level	↕	Lower abstraction level

Example:

Acceptance Criteria:
Password reset email is sent to the registered user, and the reset link expires after 15 minutes.

Test Case:
Request password reset, open the email, click the link within and after 15 minutes, and verify system behavior.

- They act as a bridge between business requirements and testing.

Handling Complex Requirements

- Complex requirements can be challenging to understand and implement.
  - ✓ Use structured approach
  - ✓ Break into components
  - ✓ Define relationships
  - ✓ Use diagrams
  - ✓ Ensure clarity

Improving Requirement Quality

Apply best practices
Follow proven techniques and guidelines to write better requirements.

Use standard templates
Use consistent templates to ensure completeness and uniformity.

Regular reviews
Review requirements with stakeholders to find issues early.

Continuous refinement
Update and improve requirements throughout the project lifecycle.

Focus on clarity
Write clear, concise, and unambiguous requirements that everyone understands.

👉 **Quality today, success tomorrow**

Summary

- Writing quality is critical for project success
- Clarity, consistency, verifiability matter
- Supports development and testing
- Reduces risks
- Enhances communication
- Ensures quality outcomes
- Acceptance criteria close the gap
- Practice improves skill

Well-written requirements build successful projects.
Keep learning. Keep practicing. Keep improving.

References

1. Beatty, K. W. (2013). Software Requirements (3rd ed.). Washington: Microsoft Press.

Thank You!