Intelligent Software Engineering

Requirements Engineering

Zhilei Ren

Dalian University of Technology

September 25, 2025

Outline

Definition of Requirements Engineering

Requirements engineering is an interdisciplinary function that mediates between the domains of the acquirer and supplier or developer to establish and maintain the requirements to be met by the system, software or service of interest. Requirements engineering is concerned with discovering, eliciting, developing, analyzing, verifying (including verification methods and strategy), validating, communicating, documenting and managing requirements[1].

[1] ISO/IEC/IEEE 29148 Systems and software engineering —Life cycle processes – Requirements engineering

As Proposed by the Project Sponsor

As Specified in the Project Request

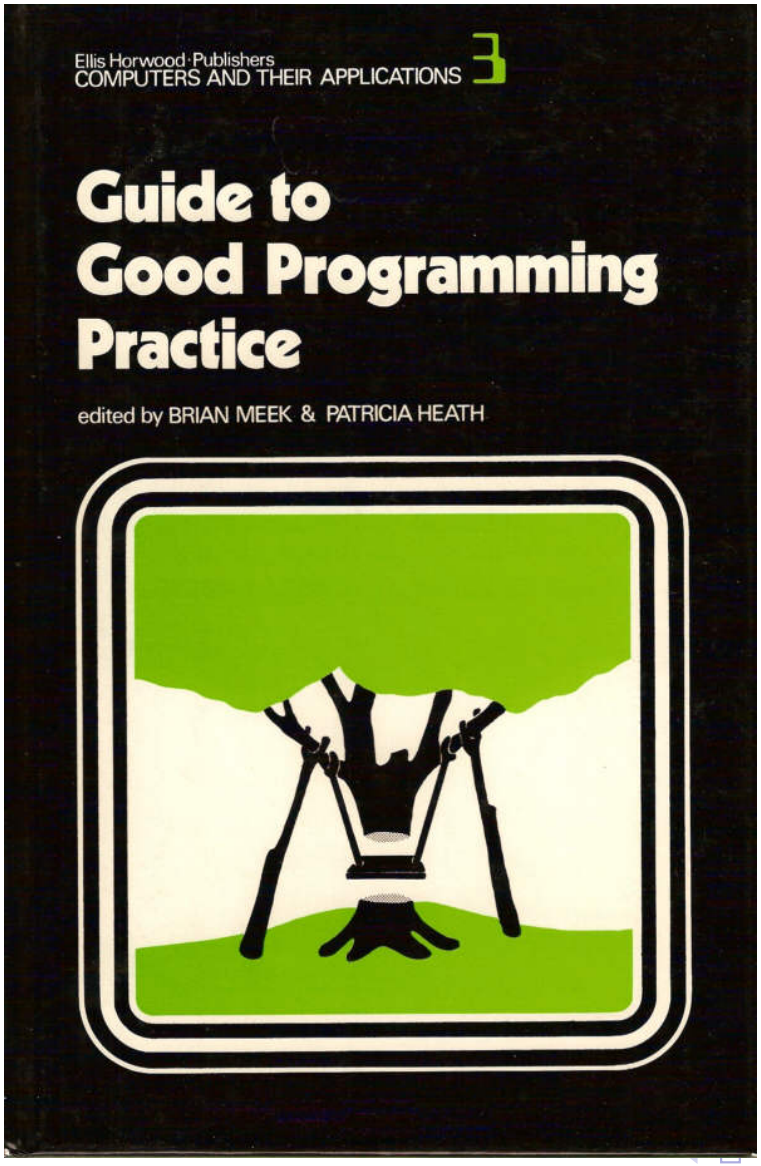
As Designed by the Senior Systems Analyst

As Produced by the Programmers

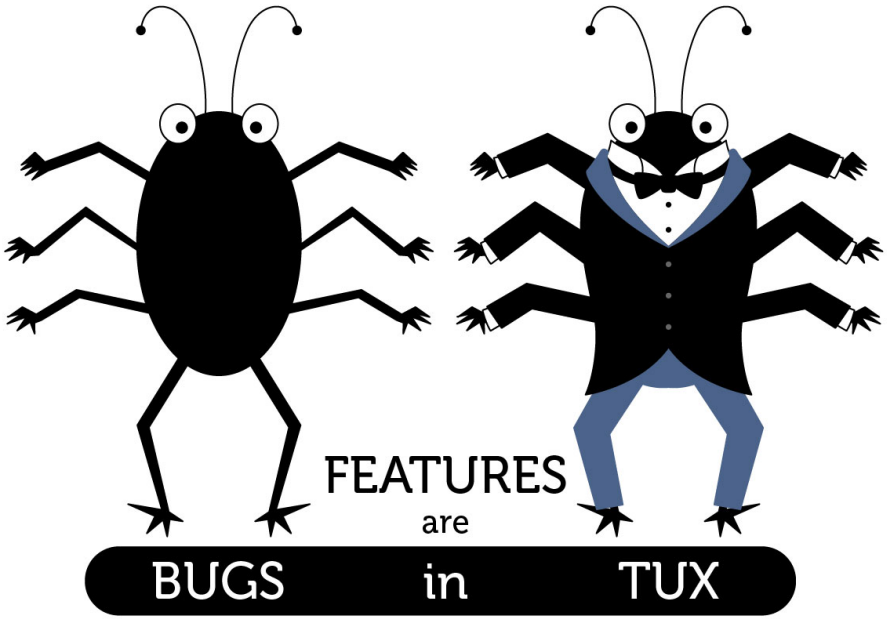
As Installed at the User's Site

What The User Wanted

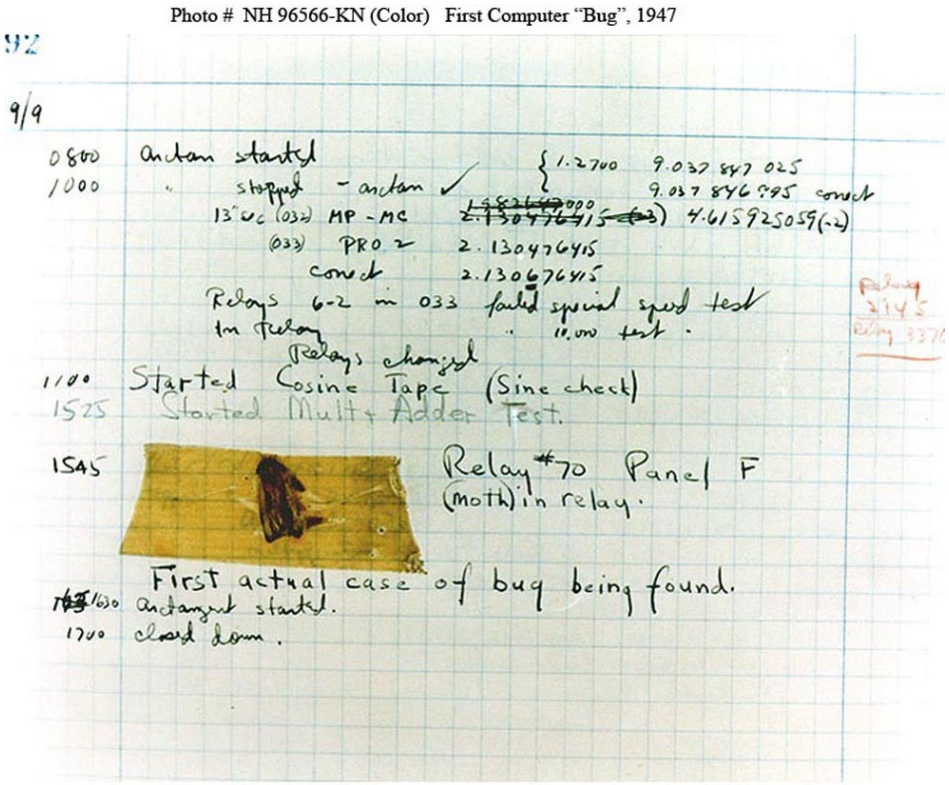
Guide to Good Programming Practice, 1979



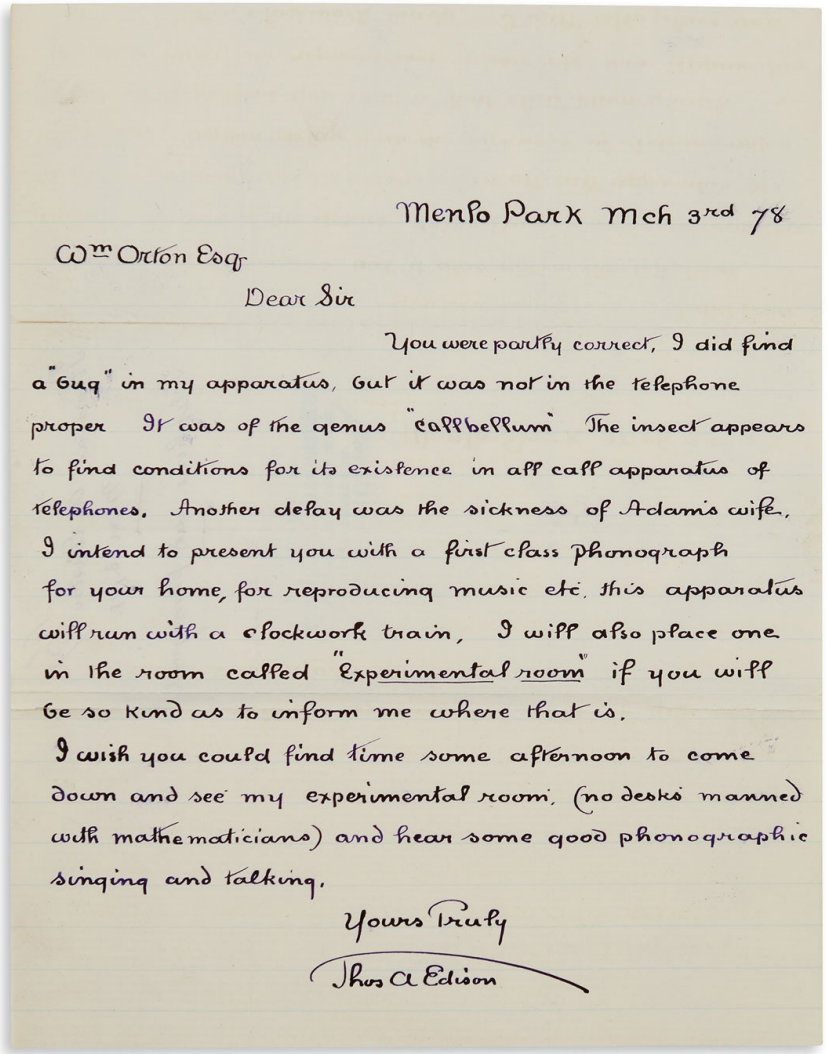
bug or feature?



The First Computer Bug



The First Computer Bug



Menlo Park Mch 3rd 78

Wm Orton Esqr

Dear Sir

You were partly correct, I did find a "bug" in my apparatus, but it was not in the telephone proper It was of the genus "Callbellum" The insect appears to find conditions for its existence in all call apparatus of telephones. Another delay was the sickness of Adams's wife. I intend to present you with a first class phonograph for your home, for reproducing music etc. this apparatus will run with a clockwork train, I will also place one in the room called "Experimental room" if you will be so kind as to inform me where that is.

I wish you could find time some afternoon to come down and see my experimental room, (no desks manned with mathematicians) and hear some good phonographic singing and talking.

Yours Truly

Thos A Edison

Combo in Fighting Games was originally a Bug

Combos were a design accident; lead producer Noritaka Funamizu noticed that extra strikes were possible during a bug check on the car-smashing bonus stage. He thought that the timing required was too difficult to make it a useful game feature, but left it in as a hidden one[2].

Figure 1: Street Fighter 2

[2] https://en.wikipedia.org/wiki/Combo_(video_games)

Hidden Features aka Easter Egg

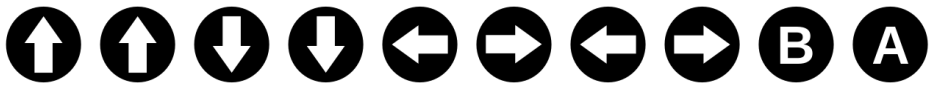


Figure 2: The Konami Code

Finding the game too difficult to play through during testing, he (Konami developer) created the cheat code, which gives the player a full set of power-ups. . . The code was meant to be removed prior to publishing, but this was overlooked and only discovered as the game was being prepared for mass production. The developers decided to leave it there, as removing it could result in new bugs and glitches[3].

[3] https://en.wikipedia.org/wiki/Konami_Code

Be Careful with Putting Easter Egg in Software

(a) Test Failure due to man

(b) The Root Cause

Figure 3: Bug Caused by an Easter Egg

Outline

What is Requirements Engineering?

Definition

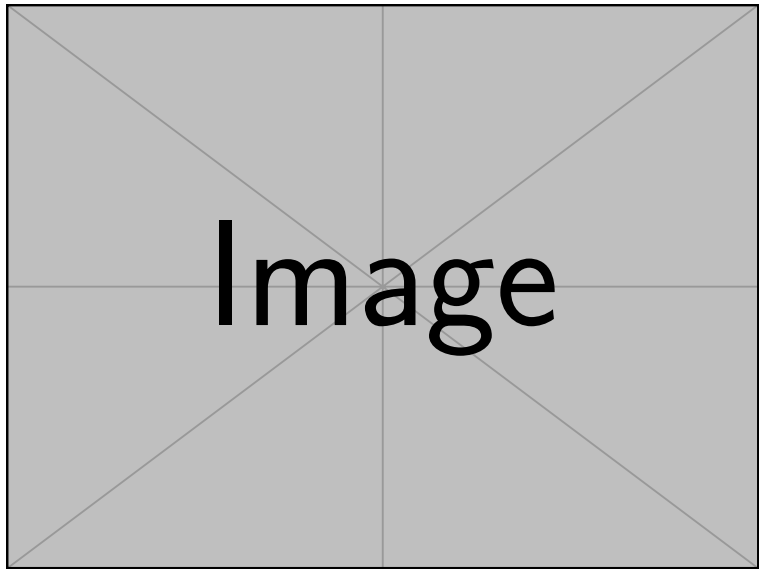
Requirements Engineering (RE) is the process of defining, documenting, and maintaining requirements in the engineering design process.

- **Elicitation**: Discovering requirements from stakeholders
- **Analysis**: Refining and modeling requirements
- **Specification**: Documenting requirements unambiguously
- **Validation**: Ensuring requirements meet stakeholder needs
- **Management**: Managing requirements changes throughout project lifecycle

Traditional RE Challenges

Common Issues:

- Incomplete requirements
- Changing requirements
- Communication gaps
- Ambiguous specifications
- Stakeholder conflicts



What is PlantUML?

- **Core Idea:** Uses simple, human-readable Domain Specific Language (DSL)
- **Foundation:** Java-based tool for layout
- **Philosophy:** Focus on content rather than manual layout

  "PlantUML is a versatile component for quickly and directly creating diagrams."

Key Advantages

Advantage	**Description**
Version Control Friendly	Text files work with Git - enables change history, diffing, collaboration
Efficiency & Speed	Faster than manual graphical editing, especially for complex diagrams
Maintainability & Consistency	Easy updates and consistent styling with themes
Automation & Integration	Integrates with documentation pipelines, build systems, CI/CD

UML Diagrams Supported

- Sequence Diagram
- Use Case Diagram
- Class Diagram
- Activity Diagram
- Component Diagram
- Deployment Diagram
- State Diagram
- Object Diagram

Visual Example:

[Diagram placeholder]

Beyond UML Diagrams

- Architectural Diagrams (C4 model)
- Entity Relationship Diagrams (ERD)
- Wireframes / UI Mockups (salt library)
- Gantt charts for project management
- Mind Maps for brainstorming
- JSON/YAML visualization
- Network diagrams

How PlantUML Works

1. **Write:** Create text file (.puml) with PlantUML syntax
2. **Process:** Java processor parses text
3. **Render:** Layout engine generates final image
4. **Output:** Get image in desired format (PNG, SVG, etc.)

Text $\rightarrow$ **PlantUML** $\rightarrow$ **Diagram**

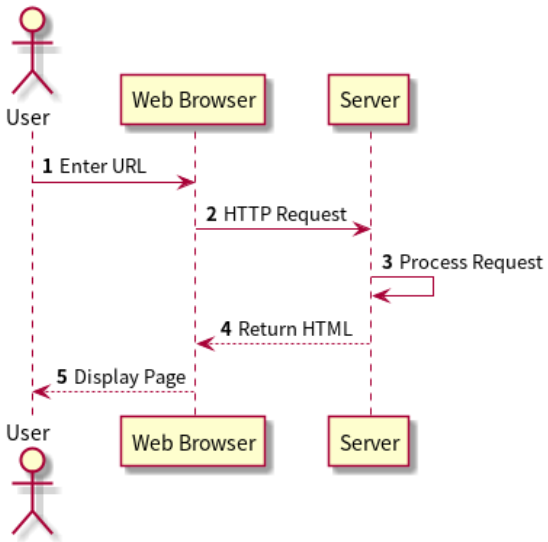
Syntax Example: Sequence Diagram

```
@startuml
actor User
participant "Web Browser" as Browser
participant Server

autonumber
User -> Browser: Enter URL
Browser -> Server: HTTP Request
Server -> Server: Process Request
Server --> Browser: Return HTML
Browser --> User: Display Page
@enduml
```

- `@startuml/@enduml`: Diagram boundaries
- `actor`, `participant`: Element declarations
- `->`, `-->`: Solid/dashed arrows
- `autonumber`: Automatic message numbering

Syntax Example: Sequence Diagram

```
@startuml
actor User
participant "Web Browser" as Browser
participant Server

autonumber
User -> Browser: Enter URL
Browser -> Server: HTTP Request
Server -> Server: Process Request
Server --> Browser: Return HTML
Browser --> User: Display Page
@enduml
```



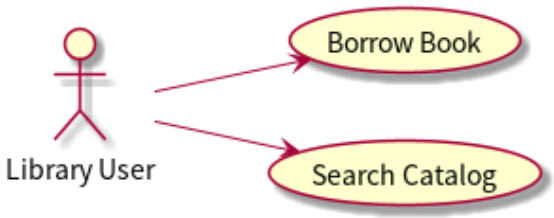
Syntax Example: Use Case Diagram

```
@startuml
left to right direction
actor "Library User" as User
usecase "Borrow Book" as Borrow
usecase "Search Catalog" as Search

User --> Borrow
User --> Search
@enduml
```

- `left to right direction`: Layout control
- `actor`, `usecase`: Actor and use case definitions
- `->`: Connection arrows

Syntax Example: Use Case Diagram

```
1 @startuml
2 left to right direction
3 actor "Library User" as User
4 usecase "Borrow Book" as Borrow
5 usecase "Search Catalog" as Search
6
7 User --> Borrow
8 User --> Search
9 @enduml
```



Getting Started with PlantUML

Online Servers (Quick Start):

- Official server: plantuml.com
- Community site: planttext.com
- No installation required

Local Installation (Recommended):

- Prerequisites: Java JRE + Graphviz
- IDE Plugins: VSCode, IntelliJ, Eclipse
- Command Line: Use `plantuml.jar`

Summary & Key Takeaways

- **Why PlantUML?** Version control, automation, efficiency
- **Text-Based:** Simple, readable language for diagrams
- **Wide Support:** Comprehensive UML + additional diagram types
- **Easy Integration:** Fits modern development workflows
- **Quick Start:** Online editors local integration

Embrace efficient diagram creation and maintenance!

What is Prompt Engineering?

Definition

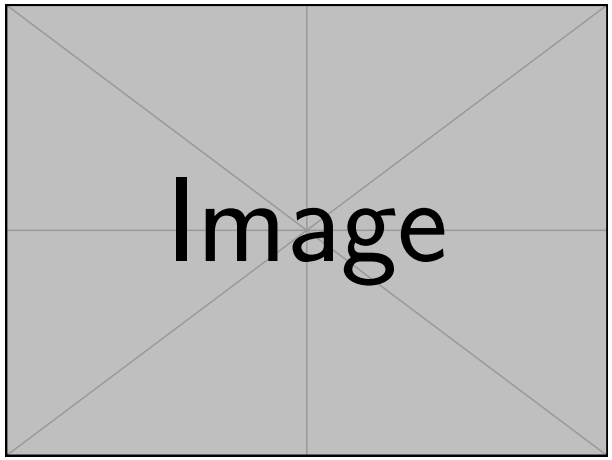
Prompt Engineering is the practice of designing and optimizing inputs (prompts) to effectively communicate with AI models to generate desired outputs.

- **Precision**: Crafting clear, specific instructions
- **Context**: Providing relevant background information
- **Structure**: Organizing information logically
- **Iteration**: Refining based on model responses
- **Validation**: Ensuring output meets requirements

Prompt Engineering Techniques

Key Techniques:

- Zero-shot vs. Few-shot prompting
- Chain-of-Thought reasoning
- Role-playing scenarios
- Template-based prompts
- Iterative refinement



Zero-shot vs. Few-shot Prompting

Zero-shot Prompting

Providing no examples - the model must understand and respond based solely on the instruction

RE Example - Zero-shot

```
"Generate functional requirements for a user login system"
```

Few-shot Prompting

Providing several examples to demonstrate the desired pattern or format

RE Example - Few-shot

```
"Here are examples of good requirements:
1.  The system shall validate user credentials
2.  The system shall encrypt passwords
```

Zero-shot vs. Few-shot: Practical Examples

Zero-shot Example:

```
Create a use case for
"password reset" functionality
```

Pros:

- Quick and simple
- Good for straightforward tasks
- Less context needed

Few-shot Example:

```
Example 1: Use case for login
- Actor: User
- Precondition: Valid account
- Steps: Enter credentials...

Example 2: Use case for logout
- Actor: User
- Precondition: Logged in
- Steps: Click logout...

Now create for "password reset"
```

Pros:

- Better quality output
- Consistent formatting
- Handles complex patterns

Chain-of-Thought Reasoning

Definition

Breaking down complex problems into intermediate reasoning steps, mimicking human problem-solving

RE Application

Instead of asking for complete requirements, guide the AI through logical steps:

1. Identify stakeholders
2. Define high-level goals
3. Break down into features
4. Specify detailed requirements
5. Validate completeness

Benefits for Requirements Engineering:

- Reduces ambiguity in complex domains

Chain-of-Thought: Requirements Example

```
Let's design requirements for an online payment system:

Step 1: Identify main stakeholders
- Customers, merchants, payment processors, banks

Step 2: Define core functionality needed
- Payment processing, security, fraud detection

Step 3: For payment processing, consider:
- Payment methods (credit card, digital wallets)
- Transaction flow (initiation, validation, completion)
- Error handling (declined payments, timeouts)

Step 4: Security requirements:
- Data encryption, PCI compliance, authentication

Step 5: Now generate detailed functional requirements
for the payment processing component...
```

Listing 1: Chain-of-Thought for Payment System

Role-playing Scenarios

Concept

Asking the AI to adopt a specific persona or role to generate more context-appropriate responses

RE Roles Examples

- **Product Owner**: "Act as a product owner for an e-commerce platform..."
- **System Architect**: "You are a system architect designing a microservices architecture..."
- **Quality Assurance Engineer**: "As a QA engineer, create test scenarios for..."
- **End User**: "From the perspective of a novice user, what features would you need..."

Advantages:

Role-playing in Requirements Elicitation

Without Role-playing:

```
List requirements for
a project management tool
```

With Role-playing:

```
Act as a project manager with
10 years experience in agile
environments. What requirements
would you prioritize for a
project management tool?
```

Specific Role Examples:

```
Role: Security Auditor
"Identify security requirements for a healthcare app handling patient data, considering HIPAA
     compliance"

Role: Accessibility Specialist
"Generate accessibility requirements for a banking application to ensure WCAG 2.1 AA compliance
     "
```

Template-based Prompts

Definition

Using structured templates to ensure consistent, comprehensive outputs across different requirements

Common RE Templates:

- Use Case Template
- User Story Template
- Functional Requirement Template
- Non-functional Requirement Template
- Acceptance Criteria Template

Benefits

- Standardized format across the project
- Easier validation and review
- Better integration with RE tools
- Consistent quality of generated content

Template Examples for Requirements Engineering

```
USE CASE TEMPLATE:
[System Name] - [Use Case Name]

ID: [Unique Identifier]
Primary Actor: [Role]
Secondary Actors: [Optional Roles]

Preconditions:
- [Condition 1]
- [Condition 2]

Main Success Scenario:
1. [Step 1]
2. [Step 2]
3. [Step 3]

Alternative Flows:
- [A1]: [Description]
- [A2]: [Description]

Postconditions:
- [State 1]
- [State 2]

Exceptions:
- [E1]: [Error condition]
```

Listing 2: Use Case Template

Functional Requirements Template

```
REQUIREMENT TEMPLATE:

Requirement ID: [FR-001]
Type: [Functional/Non-functional]
Priority: [High/Medium/Low]

Description:
[Clear, unambiguous description of what the system shall do]

Source:
[Stakeholder/Business Need/Regulation]

Rationale:
[Why this requirement is needed]

Acceptance Criteria:
1. [Criterion 1 - testable condition]
2. [Criterion 2 - testable condition]
3. [Criterion 3 - testable condition]

Dependencies:
- [Related requirement IDs]

Constraints:
- [Technical/business constraints]

Status: [Proposed/Approved/Implemented]
```

Listing 3: Structured Requirements Template

Iterative Refinement

Concept

Starting with broad requirements and progressively refining them through multiple iterations of feedback and clarification

Iterative Process:

1. **First Pass**: High-level requirements and scope
2. **Second Pass**: Detailed functional requirements
3. **Third Pass**: Non-functional requirements and constraints
4. **Final Pass**: Validation and acceptance criteria

RE Application

- Start with epic-level user stories
- Break down into features
- Refine into detailed user stories
- Add technical specifications
- Validate with stakeholders

Iterative Refinement Example

Iteration 1: High-level

```
Create requirements for
user authentication system
```

Iteration 2: More specific

```
Add multi-factor
authentication support
```

Iteration 3: Detailed

```
Specify timeout policies
and password complexity rules
```

Iteration 4: Validation

```
Add error handling for
failed authentication attempts
```

Benefits of Iterative Approach:

- Catches ambiguities early
- Allows for stakeholder feedback at each stage
- Reduces rework
- Builds comprehensive requirement sets

Complete Iterative Refinement Workflow

```
// Stage 1: Scope Definition
"Define the high-level scope for a customer relationship management system"

// Stage 2: Feature Identification
"Based on the scope, identify key features needed for lead management"

// Stage 3: Detailed Requirements
"For the lead management feature, specify functional requirements including lead capture,
     assignment, and tracking"

// Stage 4: Validation Criteria
"Add acceptance criteria and test scenarios for lead assignment functionality"

// Stage 5: Edge Cases
"Identify and specify requirements for edge cases like duplicate lead detection and lead
     expiration"
```

Listing 4: Multi-stage Refinement Process

Technique Selection Guide

Technique	Best For	RE Phase
Zero-shot	Simple, well-understood requirements	Initial scoping
Few-shot	Consistent formatting, complex patterns	Detailed specification
Chain-of-Thought	Complex domains, logical decomposition	Analysis and modeling
Role-playing	Stakeholder perspective, conflict resolution	Elicitation and validation
Template-based	Standardization, tool integration	Specification and documentation
Iterative refinement	Complex systems, evolving requirements	All phases, especially management

Table 1: Selecting the Right Prompt Engineering Technique

Combination Approach:

- Start with role-playing for stakeholder perspective
- Use chain-of-thought for complex analysis
- Apply templates for consistent documentation
- Iterate based on feedback

Case Study: Applying Multiple Techniques

Scenario: Healthcare Patient Portal

Developing requirements for a secure patient portal with medical records access

Applied Techniques:

1. **Role-playing**: "Act as a healthcare compliance officer concerned with HIPAA"
2. **Chain-of-Thought**: Break down security requirements step by step
3. **Template-based**: Use standardized requirement templates
4. **Few-shot**: Provide examples of good medical software requirements
5. **Iterative refinement**: Multiple rounds of stakeholder review

Results:

- 30% more comprehensive security requirements
- Early identification of compliance gaps
- Consistent documentation format

Summary: Prompt Engineering for RE

Key Takeaways:

- Different techniques serve different RE needs
- Techniques can and should be combined
- Start simple, add complexity as needed
- Always validate AI-generated requirements
- Document your prompt strategies for reproducibility

Next Steps:

- Practice each technique with sample RE scenarios
- Develop organizational prompt templates
- Establish validation processes for AI-generated content
- Integrate with existing RE workflows

Outline

Parallels Between RE and Prompt Engineering

Requirements Engineering	**Prompt Engineering**
Stakeholder needs analysis	Understanding user intent
Requirements elicitation	Prompt design and formulation
Use case modeling	Scenario specification
Requirements specification	Clear instruction crafting
Validation and verification	Output evaluation and refinement
Change management	Prompt iteration

Table 2: Comparative Analysis

AI-Assisted Requirements Engineering

The New Paradigm

Using prompt engineering techniques to enhance traditional requirements engineering processes

- **Automated Requirements Generation**: AI can help draft initial requirements
- **Requirements Analysis**: AI can identify inconsistencies and gaps
- **Use Case Development**: AI can generate and refine use cases
- **Specification Writing**: AI can help create clear, unambiguous specifications
- **Validation Support**: AI can simulate scenarios and identify edge cases

Outline

Sample Prompt for Use Case Generation

Effective Prompt Structure

```
Act as a requirements engineer. Generate a use case
diagram for an online banking system including:
authentication, balance check, funds transfer, and
transaction history. Use PlantUML syntax.
```

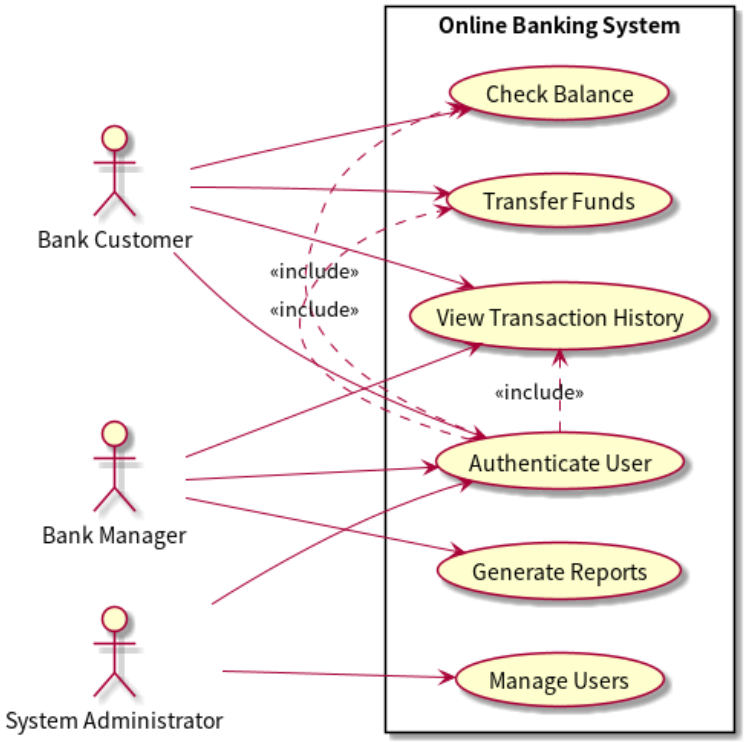
- **Role specification**: "Act as a requirements engineer"
- **Context**: "online banking system"
- **Specific requirements**: listed functionalities
- **Format specification**: "Use PlantUML syntax"

Generated Use Case Diagram (PlantUML Code)

```
@startuml
left to right direction
actor "Bank Customer" as Customer
actor "Bank Manager" as Manager
actor "System Administrator" as Admin

rectangle "Online Banking System" {
    usecase "Authenticate User" as UC1
    usecase "Check Balance" as UC2
    usecase "Transfer Funds" as UC3
    usecase "View Transaction History" as UC4
    usecase "Generate Reports" as UC5
    usecase "Manage Users" as UC6
}
Customer --> UC1
Customer --> UC2
Customer --> UC3
Customer --> UC4
Manager --> UC1
Manager --> UC4
Manager --> UC5
Admin --> UC1
Admin --> UC6
UC1 .> UC2 : <<include>>
UC1 .> UC3 : <<include>>
UC1 .> UC4 : <<include>>
@enduml
```

Listing 5: Online Banking System Use Case Diagram

Expected Use Case Diagram Output



Key Components:

- Actors: Customer, Manager, Administrator
- Use Cases: Authentication, Balance Check, Funds Transfer, etc.
- Relationships: Associations and «include» dependencies

Use Case Specification Prompt

Detailed Use Case Prompt

```
Expand the 'Transfer Funds' use case with:
primary actor, preconditions, main success scenario,
alternative flows, and postconditions. Format as a
structured use case specification.
```

Expected Output Structure:

- Use Case Name and ID
- Primary Actor
- Preconditions
- Main Success Scenario (steps)
- Alternative Flows
- Postconditions

Generated Use Case Specification

```
Use Case: Transfer Funds
ID: UC-003
Primary Actor: Bank Customer

Preconditions:
1. User is authenticated
2. User has at least one active account
3. Sufficient funds available for transfer

Main Success Scenario:
1. User selects "Transfer Funds" option
2. System displays transfer form
3. User enters recipient details and amount
4. System validates transfer details
5. System processes transfer
6. System displays confirmation
7. System updates transaction history

Alternative Flows:
A1: Insufficient funds - System shows error
A2: Invalid recipient - System validates and rejects

Postconditions:
1. Funds deducted from source account
2. Funds credited to recipient account
3. Transaction recorded in history
```

Listing 6: Transfer Funds Use Case Specification

Outline

From Use Cases to Formal Specifications

Specification Generation Prompt

```
Convert the 'Transfer Funds' use case into functional
requirements specifications. Include: unique ID,
description, priority, acceptance criteria, and
dependencies.
```

Specification Components:

- Requirement ID (e.g., FR-001)
- Description (clear, unambiguous)
- Priority (High/Medium/Low)
- Acceptance Criteria (testable conditions)
- Dependencies (related requirements)

Sample Requirements Specification Output

```
Requirement ID: FR-001
Description: The system shall allow authenticated
users to transfer funds between their own accounts
or to other registered users.

Priority: High

Acceptance Criteria:
1. User must be successfully authenticated
2. Source account must have sufficient funds
3. Transfer amount must be positive numerical value
4. Recipient account must be valid and active
5. System must display transfer confirmation
6. Transaction must be recorded in audit log
7. Both accounts must reflect updated balances

Dependencies:
- FR-010: User Authentication
- FR-011: Account Balance Check
- FR-012: Account Validation

Status: Approved
Version: 1.0
```

Listing 7: Functional Requirements Specification

Non-Functional Requirements Example

```
Requirement ID: NFR-005
Description: The funds transfer functionality
shall meet specific performance and security standards.

Category: Performance & Security

Performance Requirements:
- Response time for transfer processing: < 2 seconds
- System availability: 99.9% uptime
- Maximum concurrent users: 10,000

Security Requirements:
- All transfers must be encrypted (TLS 1.2+)
- Two-factor authentication for transfers > $1,000
- Audit trail maintained for 7 years
- Fraud detection mechanisms in place

Compliance: PCI DSS, GDPR, SOX

Testing: Load testing, security penetration testing
```

Listing 8: Non-Functional Requirements

Outline

Prompt Engineering Patterns for RE

Effective Patterns:

- **Template-based**: Use structured templates
- **Iterative refinement**: Build complexity gradually
- **Context provision**: Include domain knowledge
- **Example-driven**: Provide similar examples
- **Validation requests**: Ask for verification

Avoid:

- Vague or ambiguous terms
- Overly complex single prompts
- Assumed domain knowledge
- Open-ended without constraints
- Inconsistent terminology

Effective Prompt Template

```
[ROLE] Act as a [specific role, e.g., senior requirements engineer]

[CONTEXT] For a [domain/system type, e.g., healthcare management system]

[TASK] Generate [specific artifact, e.g., use case diagram]

[REQUIREMENTS] That includes:
- [Feature 1, e.g., patient registration]
- [Feature 2, e.g., appointment scheduling]
- [Feature 3, e.g., medical records access]

[CONSTRAINTS] With the following constraints:
- [Constraint 1, e.g., HIPAA compliance]
- [Constraint 2, e.g., mobile-first design]

[FORMAT] Present the output in [format, e.g., PlantUML syntax]

[VALIDATION] Also include [validation request, e.g., acceptance criteria]
```

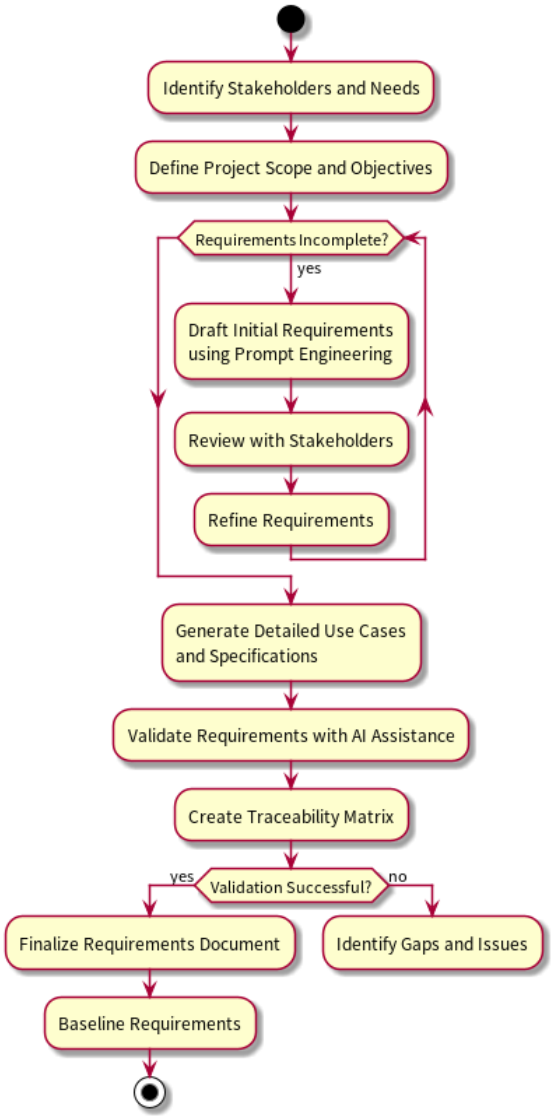
Listing 9: Structured Prompt Template for RE

RE-PE Integration Workflow

```
@startuml
start
:Identify Stakeholders and Needs;
:Define Project Scope and Objectives;

while (Requirements Incomplete?) is (yes)
  :Draft Initial Requirements\nusing Prompt Engineering;
  :Review with Stakeholders;
  :Refine Requirements;
endwhile

:Generate Detailed Use Cases\nand Specifications;
:Validate Requirements with AI Assistance;
:Create Traceability Matrix;

if (Validation Successful?) then (yes)
  :Finalize Requirements Document;
  :Baseline Requirements;
  stop
else (no)
  :Identify Gaps and Issues;
  detach
endif
@enduml
```

Listing 10: Requirements Engineering Workflow

RE-PE Integration Workflow



Outline

Prompt Version Control Example

```
# Prompt Template v2.1 - Use Case Generation
# Date: 2024-01-15 | Author: RE Team
# Changelog: Added validation criteria

SYSTEM_PROMPT: """
You are an experienced requirements engineer
specializing in [DOMAIN]. Your task is to create
comprehensive use case specifications.
"""

USER_PROMPT: """
Generate a use case for [FEATURE] in [SYSTEM] with:

1. PRIMARY_ACTOR: [actor role]
2. PRECONDITIONS: [system state requirements]
3. MAIN_SCENARIO: [step-by-step flow]
4. ALTERNATIVE_FLOWS: [error handling paths]
5. POSTCONDITIONS: [end state conditions]
6. VALIDATION: [acceptance criteria]

Format: Structured text with clear sections.
"""
```

Listing 11: Version-Controlled Prompt Template

Case Study: E-commerce System

Project Overview

Developing an e-commerce platform using AI-assisted requirements engineering

Results:

- 40% reduction in initial requirements gathering time
- 25% fewer requirements defects identified during testing
- Improved stakeholder satisfaction with visual models
- Better traceability through structured prompts

Challenges:

- Training team on effective prompt engineering
- Integrating AI outputs with existing workflows
- Ensuring consistency across multiple AI sessions

Outline

Key Takeaways

- **Synergy**: Prompt Engineering enhances traditional Requirements Engineering
- **Efficiency**: AI can accelerate requirements development and validation
- **Quality**: Structured prompts lead to better requirements specifications
- **Accessibility**: Visual models (like PlantUML) improve stakeholder understanding
- **Iterative Nature**: Both RE and PE benefit from continuous refinement

The Future

AI will become an integral partner in requirements engineering, with prompt engineering as the essential interface between human intent and machine capability.

References and Further Reading

[1] Sommerville, I. (2011) Software Engineering

[2] Wiegers, K. & Beatty, J. (2013) Software Requirements

[3] White, J. et al. (2023) A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT

[4] PlantUML Documentation (2024) Use Case Diagram Guide

[5] Nuseibeh, B. & Easterbrook, S. (2000) Requirements Engineering: A Roadmap

Thank You!

Questions?