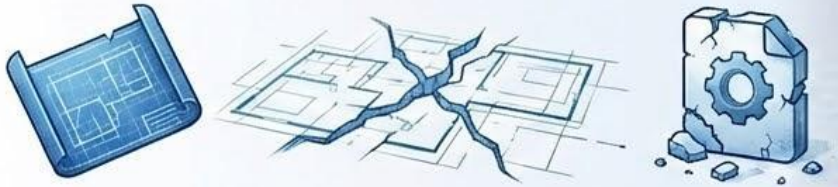
Chapter 5: Understanding Requirements

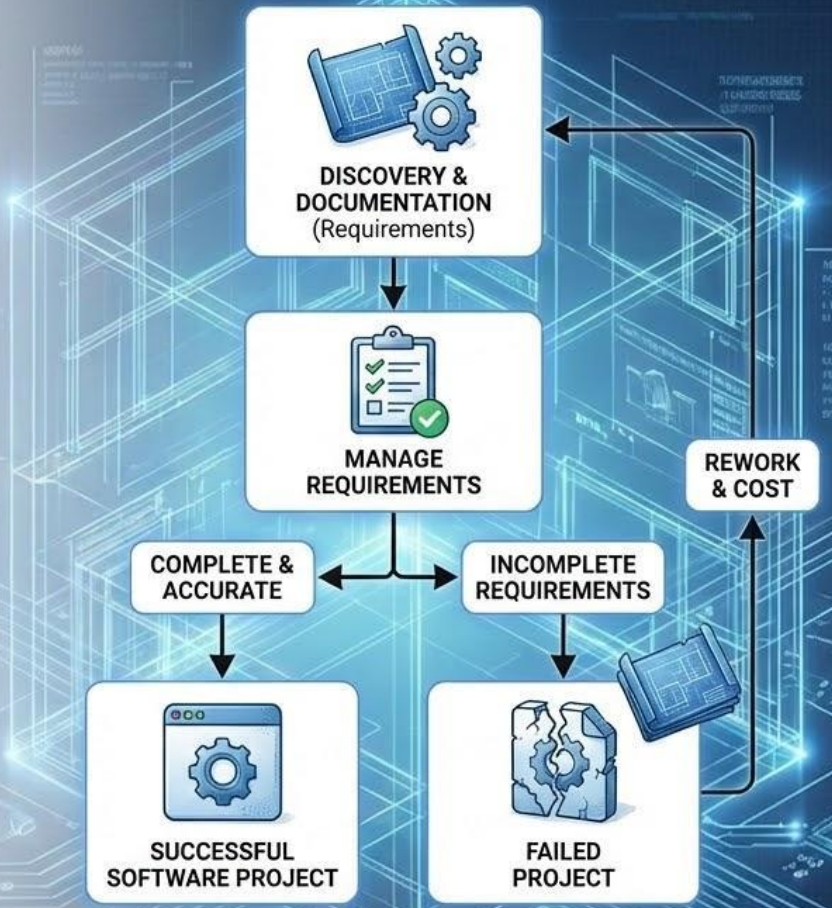
THE COST OF ERROR



The #1 cause of failed software projects is incomplete requirements.

Building the *right software* is harder than building the *software right*.

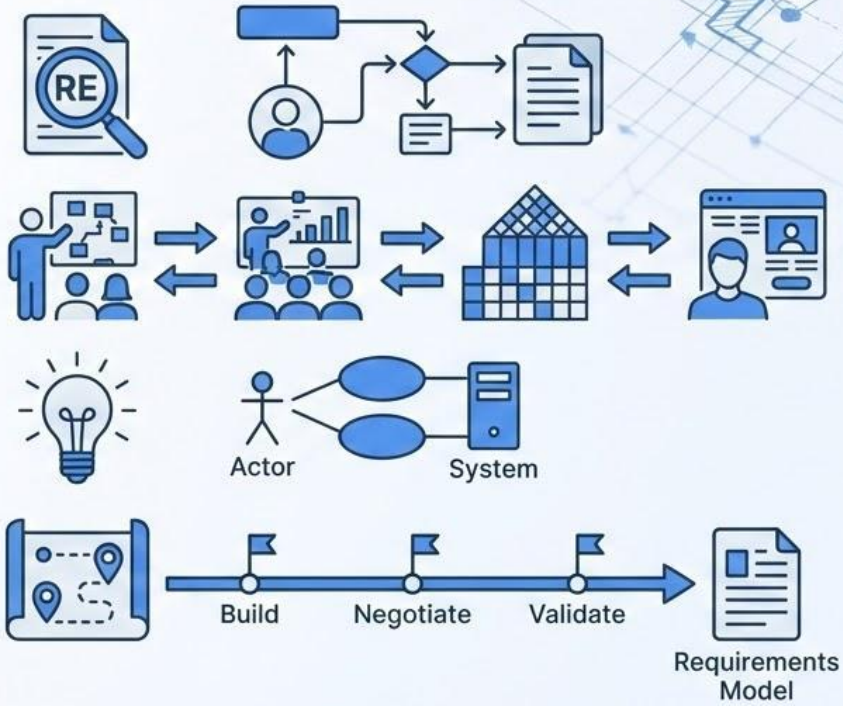
Today, we learn **Requirements Engineering:** the discipline of discovering, documenting, and managing what the system must do.



Lecture Objectives

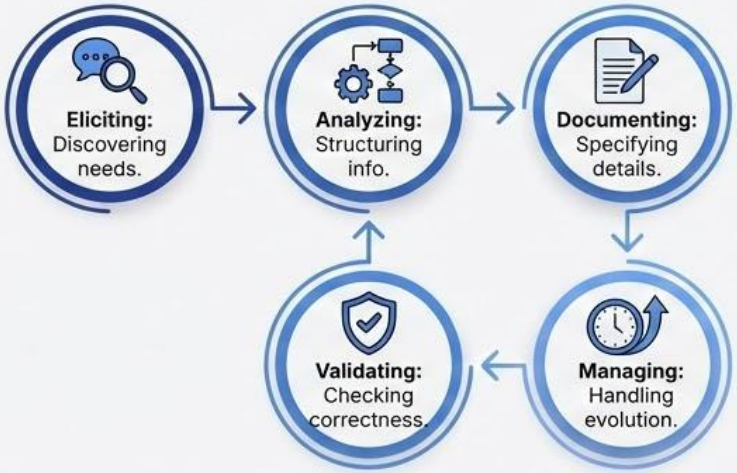
By the end of this lecture:

- **Define** Requirements Engineering and explain its importance in the software lifecycle.
- **Describe** the steps for establishing groundwork: stakeholder identification and collaboration.
- **Compare** techniques for eliciting requirements: collaborative gathering, QFD, and usage scenarios.
- **Explain** the purpose and structure of Use Cases.
- **Outline** the process of building, negotiating, and validating a requirements model.



Fundamentals of Requirements Engineering

The 5 Activities



Types of Requirements

Functional: What the system does.
(e.g., Calculate expenses)

Non-Functional (NFRs): Constraints/Quality.
(e.g., Performance, Security)

Business Rules: Policies.
(e.g., Data encryption laws)

The RE Process Flow

1. **Inception:** Define the scope and nature of the problem.
2. **Elicitation:** Gather requirements from stakeholders.
3. **Elaboration:** Create analysis models (Modeling).
4. **Negotiation:** Agree on a deliverable system.
5. **Specification:** Document the requirements formally.
6. **Validation:** Ensure quality and correctness.
7. **Management:** Control changes over time.

Identifying Stakeholders

"Miss a key stakeholder, miss a key requirement."

Business

End-users, managers, and subject matter experts who define the business need.

Technical

System admins, developers, and support staff who maintain the solution.

External

Regulators, customers, and partners who influence constraints.

Multiple Viewpoints & Collaboration

Conflicting Viewpoints

- **User:** Ease of use and efficiency.
- **Manager:** Reporting and cost control.
- **Admin:** Maintenance and security.

Challenge: Reconcile these into a coherent set.

Collaboration

RE is a team sport. It requires a collaborative partnership between developers and customers.

Techniques:

- Facilitated Workshops (JAD)
- Brainstorming Sessions

Asking the First Questions

Scoping the Problem

Before detailed elicitation, establish the context:

- **Source:** Who is behind the request?
- **User:** Who will use the solution?
- **Benefit:** What is the business value? (The “Why”)
- **Constraints:** Budget, timeline, technology limits?
- **Interfaces:** What other systems are involved?

Elicitation Techniques

Collaborative Gathering

Facilitated workshops (JAD) involving all stakeholders.

- **Preparation**: Identify agenda.
- **Facilitation**: Neutral guide.
- **Outcome**: Shared understanding.

Usage Scenarios

Narrative descriptions of user interaction.

Mary, a student, logs into the portal

clicks 'Course Registration', searches for 'SE500'

and adds it to her cart.

Purpose: Makes abstract requirements concrete.

Quality Function Deployment (QFD)

A technique to translate **Customer Needs** ("Voice of Customer") into **Technical Requirements**.

The House of Quality

- **Left Wall:** Customer requirements (e.g., "Fast").
- **Ceiling:** Technical characteristics (e.g., "Response time").
- **Middle:** Relationship matrix showing impact.

Goal: Maximize customer satisfaction through engineering priority.

Use Cases

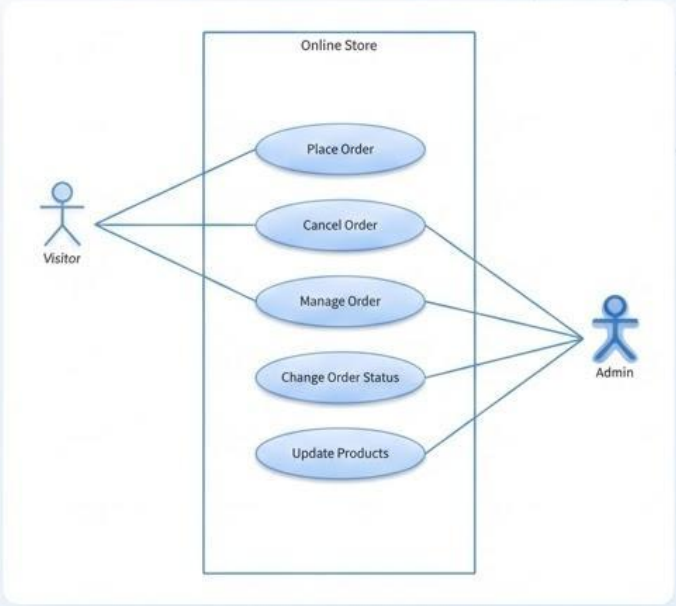
A structured version of a scenario describing a discrete unit of functionality.

Key Components

Actor: A role (person/system) interacting with the system.

Main Flow: Step-by-step success scenario.

Alternate Flows: Error handling and branches.



The raw outputs of the elicitation process:

Lists

Requirements, features, and constraints.

Notes

From meetings and stakeholder interviews.

Records

Audio recordings and transcripts.

Sketches

Preliminary prototypes and mockups.

Developing Use Cases

A formal, structured documentation of system behavior.

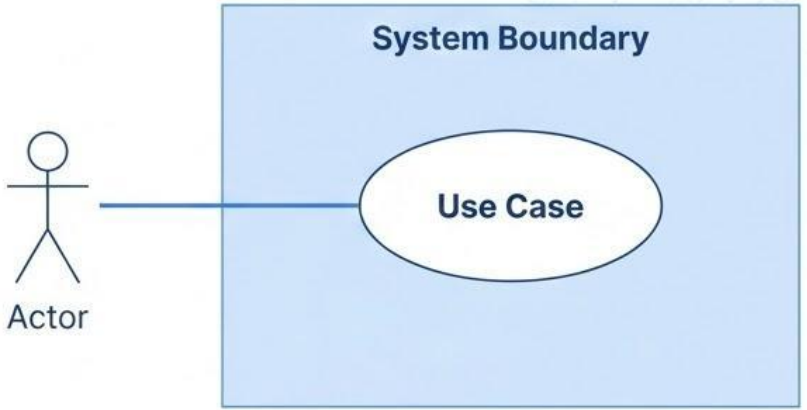
Components & Value

Actor: Entity interacting with system.

Use Case: Unit of functionality.

Description: Preconditions, Main Flow, Extensions.

Value:
Foundation for system design & test cases.



Building the Requirements Model

Multiple views to create a complete picture.

Scenario-Based

Use Cases & User Stories.
(User perspective)

Class-Based

UML Class Diagrams.
(Data & Logic)

Behavioral

State & Sequence Diagrams.
(Response to events)

Flow-Oriented

Data Flow Diagrams.
(Data transformation)

Analysis Patterns

Concept

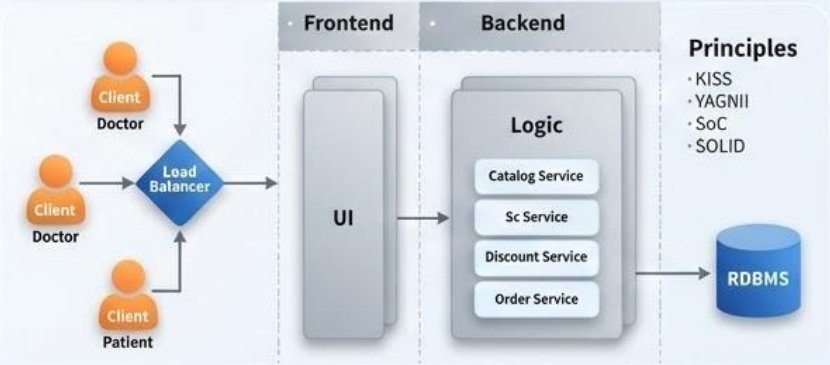
Reusable models for common problems within a specific application domain.

Key Benefit

Speeds up modeling by applying proven solutions instead of reinventing the wheel.

Example: Healthcare "Observation"

- **Observer:** (Doctor/Nurse)
- **Measurement:** (Blood Pressure/Temp)
- **Phenomenon:** (The patient's condition)



Negotiating Requirements

Reality: Unlimited wants vs. Limited resources.

MoSCoW Prioritization

Must have
Critical for delivery.

Should have
Important but not vital.

Could have
Desirable/Nice-to-have.

Won't have
Lowest priority this time.

Goal: A negotiated specification forming the project contract.

Validating Requirements

The Quality Checklist

- **Correct:** Accurate representation of need?
- **Complete:** Are any scenarios missing?
- **Consistent:** No conflicting statements?
- **Unambiguous:** Only one interpretation?
- **Testable:** Can we prove it works?

Validation Techniques

- **Formal Reviews:** Walkthroughs.
- **Prototyping:** Mock-ups.
- **Test Case Generation:** If you can't write a test, the requirement is bad.

Conclusion & Key Takeaways

Foundational

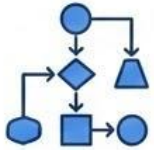
RE discovers, documents, and manages what to build.

Stakeholders

Identify everyone, foster collaboration.

Elicit Systematically

Use workshops, QFD, scenarios.



Model Formally

Use Cases and Analysis Models create precision.

Negotiate & Validate

Ensure requirements are realistic and agreed upon.

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult."

— Fred Brooks

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult."

— Fred Brooks