

Software Requirements Engineering

软件需求工程

Fahad Sabah

CONTENTS



01 Why Requirements Matter: The Bridge Between Ideas and Reality 为什么需求很重要：想法与现实之间的桥梁



02 The Requirements Universe: Types, Frameworks, and Real-World Challenges 需求范围：类型、框架和现实世界的挑战



03 The RE Toolkit: Elicitation and Documentation Techniques RE 工具包：引出和记录技术



04 Ensuring Success: Validation, Verification, and Management 确保成功：验证、验证和管理

Course Overview: What is this course?

Course Overview

A guide to:

- **Finding** what needs to be built (Elicitation).
- **Communicating** it clearly (Documentation).
- **Managing** changes and ensuring correctness (Validation & Management).

This is not about paperwork; it's about **clear thinking and effective communication.**

课程概述： 这门课程是什么？

课程概述

指南：

- 找到需要构建的东西（引出）。
- 清楚地传达它（文档）。
- 管理变更并确保正确性（验证和管理）。

这与文书工作无关;这是关于清晰的思维和有效的沟通。

Course Overview[课程概述]

课程名称/Course Title: 软件需求工程（英文）/ Software Requirements Engineering

课程代码/Course Code:

任课教师/Instructor(s): 法赫德萨巴赫/ Fahad Sabah

开课学部（院）/Faculty: 计算机学院

1. 课程概要/Course Summary				
课程名称（中文） Course Title (Chinese)	软件需求工程（英文）			
课程名称（英文） Course Title (English)	Software Requirements Engineering			
适用层次(可多选) Applicable level(s)	<input checked="" type="checkbox"/> 学硕 (Academic master)	<input type="checkbox"/> 学博 (Academic Ph.D)	<input type="checkbox"/> 专硕 (Professional master)	<input type="checkbox"/> 专博 (Professional Ph.D)
授课语言 Teaching Language	英语 English	适用学科/专业 学位类别（领域） Discipline	软件工程 Software Engineering	
学分数 Course Credit(s)	32	开课学期 Semester		
总学时 Teaching Hours in Total	共 32 学时 32 teaching hours	实验/实践学时 Hours for Experiments /Practice	共 0 学时 0 teaching hours	
预修课程要求 Pre-requisite Course(s)	软件工程导论、面向对象编程、系统分析与设计 Introduction to Software Engineering, Object-Oriented Programming, Systems Analysis and Design			

What You Will Learn (Learning Outcomes)

By the end of this course, you will be able to:

- **Explain** the role and importance of Requirements Engineering (RE) in software project success.
- **Identify** different types of requirements and key project stakeholders.
- **Create** key requirements artifacts like user stories and models.
- **Apply** techniques for eliciting, validating, and managing requirements.

您将学到什么（学习成果）

在本课程结束时，您将能够：

- 解释需求工程（RE）在软件项目成功中的作用和重要性。
- 确定不同类型的需求和关键项目利益相关者。
- 创建关键需求项目，如用户情景和模型。
- 应用技术来引出、验证和管理需求。

课程考核及成绩评定

Course Assessment & Grading

How You Will Be Assessed (Assessment Overview)

Assessment Overview

- **Group Project (Primary Focus):** Create a Software Requirements Specification (SRS) document for a sample project.
- **Quizzes/Exams:** Test your understanding of key concepts and terminology.
- **Class Participation:** Engage in discussions and activities (like the one we're about to do!).

课程考核及成绩评定

Course Assessment & Grading

考核指标* Assessment Criteria	权重 Percentage	评定标准 Assessment Standard
课堂表现 Participation	10	积极参与课堂讨论 Participation in group discussions
作业/实验 Assignment(s)	20	按时提交所有作业 Timely submission of all assignments 代码符合规范要求 Code meets specified standards 功能实现完整 Complete implementation of required functions 报告内容清晰准确 Clear and accurate reports
课程考试 Course Exam(s).	30	期末考试 (30%)：综合考核全部课程内容 Final (30%): Comprehensive assessment of all course content
项目 Project	40	项目完整性 (16%)：实现所有要求功能 Completeness (16%): All required features implemented 代码质量 (12%)：符合PEP8规范，结构清晰 Code quality (12%): PEP8 compliant, well-structured 文档 (8%)：包含清晰的README和注释 Documentation (8%): Clear README and comments 创新性 (4%)：体现创造性解决方案 Creativity (4%): Demonstrates innovative solutions

Fundamentals:

What are Software Requirements?

A condition or capability needed by a user to solve a problem or achieve an objective.

Think of a Blueprint for a House

You would never start construction without a detailed blueprint.

- The Blueprint aligns the vision of the homeowner, architect, and builder.
- It specifies what will be built (3 bedrooms, 2 bathrooms), not how to pour the foundation.

Software Requirements are the blueprint for your system.

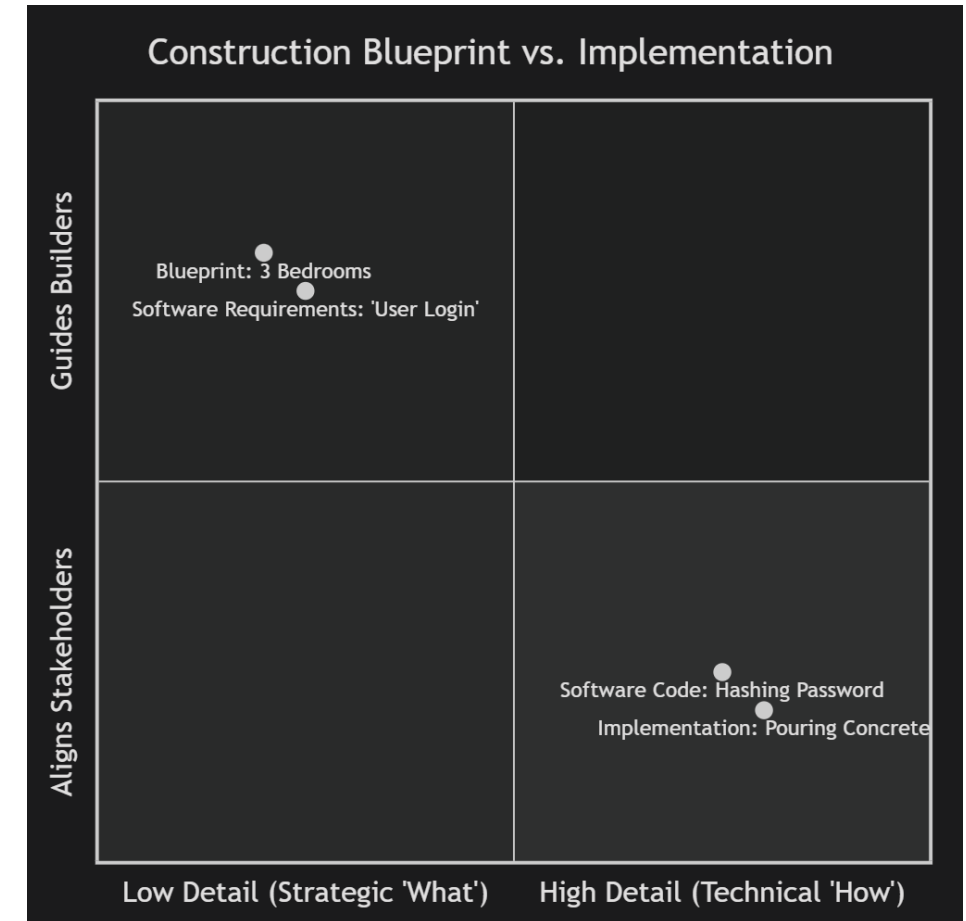
They focus on the 'What,' not the 'How.' They describe what the system must do—like 'the user must be able to log in'—not how to code it.

What it specifies (The Requirements):

- ❑ Scope: "A house with 3 bedrooms and 2 bathrooms."
- ❑ Layout: "The master bedroom should be connected to bathroom #1."
- ❑ Components: "A kitchen island with a sink."
- ❑ Constraints: "Must use bricks for the exterior."

What it doesn't specify (The Implementation):

- ❑ How to mix the concrete for the foundation.
- ❑ Which brand of nails the Carpenter should use.
- ❑ The exact sequence for framing the walls.



The Software Requirements (The "What") 软件要求 (“什么”)

It aligns the business, product owner, and development team on the what. It doesn't specify the how, like the specific algorithm to use.

What it specifies (The Requirements):

- ☐ Scope: "The system shall allow users to log in."
- ☐ Functionality: "Login shall be possible via email or username."
- ☐ Components: "Failed login attempts shall be logged for security."
- ☐ Constraints: "Passwords must be stored encrypted."

What it doesn't specify (The Implementation):

- ☐ How to hash the password (e.g., bcrypt vs. scrypt).
- ☐ The exact code for the user interface. etc.

01

Why Requirements Matter
The Bridge Between Ideas and Reality
为什么需求很重要 思想与现实之间的桥梁



需求工程是什么与为何重要

What is and why requirements engineering is important



01

需求工程的定义 (Definition of requirements engineering)

需求工程是定义、文档化和维护需求的系统过程。它确保软件开发有明确的方向和目标，避免项目偏离预期。

Requirements engineering is the systematic **process of defining, documenting, and maintaining** requirements. It ensures that software development has a clear direction and goals, avoiding projects deviating from expectations.

02

重要性 (Importance)

需求工程的重要性在于它为后续的设计、测试和验收提供了基础。前期的共识可以减少后期的返工和成本。

The importance of requirements engineering is that it provides the **basis for subsequent design, testing, and acceptance**. Consensus in the early stages can reduce rework and costs in the later stages.

缺陷发现越晚代价越高

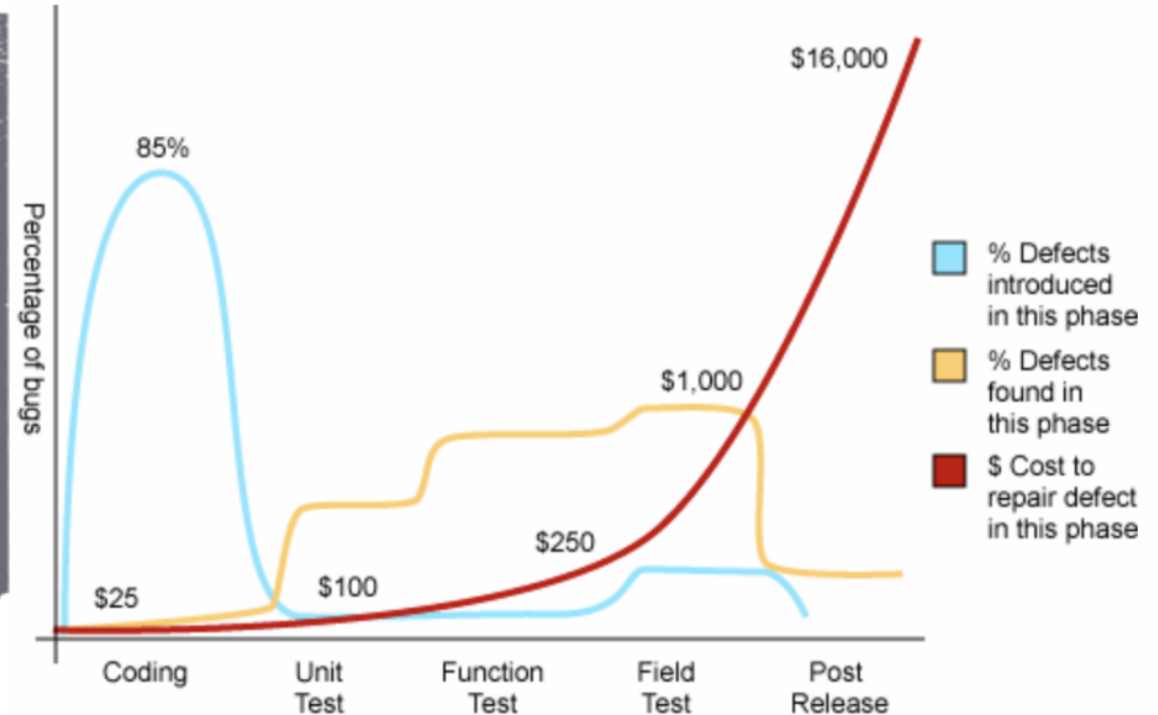
The later the defect is discovered, the more costly it is

成本与时间的关系

The relationship between cost and time

在开发周期中，缺陷发现得越晚，修复成本越高。例如，运维阶段修复需求阶段的缺陷，成本可能放大百倍。

During the development cycle, the later defects are discovered, the higher the cost of fixing. For example, repairing defects in the requirements stage during the O&M phase can increase the cost by a hundredfold.



Source: Applied Software Measurement, Capers Jones, 1996

A Tale of Two Projects

Project A: The "Perfect" Failure

- An app was built exactly to the specification.
- It was technically flawless.
- But... nobody used it.

It solved the wrong problem.

Project B: The "Simple" Success

- A small, simple feature was added to a website.
- It had a huge impact on user satisfaction and sales.

Why? It addressed a real, unspoken user need.

两个项目的故事

项目 A: “完美” 失败

- 应用完全按照规范构建。
 - 它在技术上是完美无缺的。
 - 但。。。没有人使用它。
- 它解决了错误的问题。

项目 B: “简单” 的成功

- 一个网站添加了一个小而简单的功能。
 - 它对用户满意度和销售产生了巨大影响。
- 为什么？它解决了真实的、不言而喻的用户需求。

Project A: The "Perfect" Failure

The Situation: A team building a photo-sharing app for families.

The Request from Grandma: "I want to see all the photos of my grandchildren in one place!"

The "Perfect" Solution Built: The developers created a beautiful app with:

- ❑ A central album where everyone could upload photos
- ❑ High-quality image storage
- ❑ Ability to tag people in photos
- ❑ Commenting features

项目 A: “完美” 失败

情况：一个团队为家庭构建照片共享应用程序。

奶奶的要求：“我想在一个地方看到我孙子孙女的所有照片！”

构建的“完美”解决方案：开发人员创建了一个漂亮的应用程序，其中包含：

- ❑ 每个人都可以上传照片的中央相册
- ❑ 高质量图像存储
- ❑ 能够标记照片中的人物
- ❑ 注释功能

Project A: The "Perfect" Failure

The Failure: Grandma never used it. The family photos stayed scattered across different phones and social media accounts.

Why It Failed: The developers solved the technical problem ("collect photos in one place") but missed the real human problem. Grandma didn't just want to *see* photos - she wanted them to come to her easily, without learning new technology. The app required her to remember a password, navigate a complex interface, and actively check for new photos.

The Real Problem They Should Have Solved: "How can we make photos automatically appear where Grandma already looks every day?"

项目 A: “完美” 失败

失败: 奶奶从未使用过它。家庭照片分散在不同的手机和社交媒体账户上。

失败原因: 开发人员解决了技术问题（“将照片收集到一个地方”），但错过了真正的人为问题。奶奶不仅想看照片，还想让照片轻松出现在她身边，而无需学习新技术。该应用程序要求她记住密码、浏览复杂的界面并主动检查新照片。

他们应该解决的真正问题是：

“我们如何才能让照片自动出现在奶奶每天已经看过的地方？”

Project B: The "Simple" Success

The Situation: The same team building a different feature for the same app.

The Observation: They noticed that young parents were taking photos of their kids but often forgot to share them with grandparents.

The Simple Feature They Added: A one-button option: "Share with Grandma." When pressed, it would automatically:

- Text the photo directly to Grandma's phone (which she already knew how to use)
- Include a simple caption like "From [Child's name]"
- No app required for Grandma to receive it

项目 B: “简单” 的成功

情况: 同一个团队为同一个应用程序构建不同的功能。

观察结果: 他们注意到年轻父母正在给孩子拍照，但经常忘记与祖父母分享。

他们添加的简单功能: 一键式选项: “与奶奶分享”。按下时，它会自动:

- 将照片直接发送到奶奶的手机（她已经知道如何使用）
- 添加一个简单的标题，例如 “来自[孩子的名字]”
- 奶奶无需应用程序即可接收

成功：照片共享增加了 500%。奶奶喜欢整天拍惊喜照片。父母喜欢它的简单性。

The Success: Photo sharing increased by 500%. Grandma loved getting surprise photos throughout her day. Parents loved how easy it was.

成功原因：他们解决了真正的问题：“父母忙着忘记分享，而奶奶却想让照片自然而然地来找她。他们使用奶奶已经理解的技术（短信），让父母毫不费力。

Why It Succeeded: They solved the real problem: "Parents are busy and forget to share, while Grandma wants photos to come to her naturally." They used technology Grandma already understood (text messages) and made it effortless for parents.

The Key Difference

主要区别

	Project A (Failure)	Project B (Success)
What they built	What was requested	What was actually needed
Focus	Technology features	Human behavior
Question asked	"What do you want?"	"What problem are you trying to solve?"
Result	Perfect system, unused	Simple feature, loved

底线：出色的软件可以解决人类问题，而不仅仅是技术要求。最好的功能通常来自了解人们实际做什么，而不仅仅是他们所说的他们想要什么。

The bottom line: Great software solves human problems, not just technical requirements. The best features often come from understanding what people actually do, not just what they say they want.

Two Main Types of Requirements

1. Functional Requirements

- Describe the behavior of the system - what it must *do*.
- *Example: "The system shall allow the user to log in with a username and password."*

2. Non-Functional Requirements (NFRs)

- Describe the qualities of the system - how well it must perform.
- *Example: "The login page must load in under 2 seconds." (Performance NFR)*

两种主要类型的要求

1. 功能要求

- 描述系统的行为 - 它必须做什么。
- 示例：“系统应允许用户使用用户名和密码登录。”

2. 非功能性需求 (NFR)

- 描述系统的质量 - 它必须执行多少。
- 示例：“登录页面必须在 2 秒内加载。”
(性能 NFR)

The Stakeholder Landscape

Who Cares? Understanding Stakeholders

A stakeholder is anyone who has a vested interest in the project's outcome. *If you don't know who the stakeholders are, you can't know what the requirements are.*

Activity: Identify the Stakeholders

Scenario: A university is building a new online course registration system.

1. Think:

- Individually, list all the people/groups who would care about this system.

2. Pair:

- Discuss your lists with the person next to you. Combine and refine.

3. Share:

- We'll share our combined lists as a Class.

Common Stakeholder Groups

- ❑ **Users / End-Users:** Students, Faculty, Registrars
- ❑ **Customers / Clients:** University Administration, Board of Trustees
- ❑ **Developers:** The people building the system (You!)
- ❑ **Project Managers:** Oversee the project timeline and budget.
- ❑ **Legal & Compliance:** Ensure data privacy laws (like FERPA) are followed.
- ❑ **Operations/IT Support:** The team that maintains the system after launch.

The first crucial step in RE is to identify ALL relevant stakeholders.

共同利益相关者群体

- ❑ 用户/最终用户：学生、教职员工、注册商
 - ❑ 客户/客户：大学行政部门、董事会
 - ❑ 开发人员：构建系统的人（你！）
 - ❑ 项目经理：监督项目时间表和预算。
 - ❑ 法律与合规：确保遵守数据隐私法（如 FERPA）。
- 运营/IT 支持：启动后维护系统的团队。
RE 的第一个关键步骤是确定所有相关利益相关者。

Session 1 Recap [第一节回顾]

Recap: The Critical Bridge

- Requirements define the "**what**" and align stakeholders.
- Poor requirements lead to failed projects, even with perfect code.
- They are the **blueprint** and foundation for agreement.
- We distinguish between **Functional** (what it does) and **Non-Functional** (how well it does it) requirements.
- Identifying **Stakeholders** is the essential first step.

02

The Requirements Universe
Types, Frameworks, and Real-World Challenges

需求范围
类型、框架和现实世界的挑战



A Layered Approach to Requirements

A Deeper Dive: The Requirements Taxonomy

Requirements exist at different levels of detail, from business goals to technical specs.

Level 1: Business Requirements

The "Why"

- High-level objectives of the organization or customer.
- Define the project's justification and overall goals.

Source: Typically from senior management, sponsors.

Example for our E-Commerce App:

- **"Increase mobile app sales by 20% in the next fiscal year."**
- **"Capture 15% of the market share for eco-friendly products."**

Level 2: User Requirements

The "Goals"

- Describe the tasks or goals users need to achieve with the system.
- Often written from the user's perspective.

Source: End-users, user representatives.

Example for our E-Commerce App:

- **"As a customer, I want to filter products by size and color so I can find relevant items quickly."**
- **"As a returning customer, I want to save my payment details to checkout faster."**

Level 3: Functional Requirements

The "System Behavior"

- Detailed descriptions of the software functionality that must be implemented.
- Derived from User Requirements. They specify *what* the system must do in detail.

Source: Analysts, developers, based on user needs.

Example (from the user requirement to filter):

- **"The system shall display filter options for 'Size' (S, M, L, XL) and 'Color'."**
- **"When 'Apply Filters' is clicked, the system shall display only products matching the selected criteria."**

Level 4: Non-Functional Requirements (NFRs)

The "Quality Attributes"

These define the *quality* of the system. Key categories:

- ❑ **Performance:** "Product search results must be displayed within 1.5 seconds."
- ❑ **Usability:** "A new user must be able to complete their first purchase in under 3 minutes."
- ❑ **Security:** "All payment transactions must be encrypted using TLS 1.3."
- ❑ **Reliability:** "The app must have 99.9% uptime."
- ❑ **Compatibility:** "The app must support iOS 15 and above."

Level 5: System Requirements & Constraints

The "Rules and Boundaries"

- These are often global constraints on the system.
- They can be functional or non-functional in nature.

Examples:

- "The application **must run on Android 10+** and iOS 14+." (Platform Constraint)
- "The system **must integrate with the existing inventory database.**" (Integration Constraint)
- "The app **shall comply with GDPR data protection regulations.**" (Regulatory Constraint)

Organizing the Chaos: Professional Frameworks

How do Professionals Manage This Complexity?

Answer: They use established Bodies of Knowledge.

These are **guides to best practices**, not step-by-step rulebooks.

Introduction to SEBoK & SWEBOK

SEBoK

- ❑ **Systems Engineering Body of Knowledge**
- ❑ Takes a **broader, system-wide view** (hardware, software, people, processes).
- ❑ Useful when software is part of a larger system (e.g., a car, an airplane).

SWEBOK

- ❑ **Software Engineering Body of Knowledge**
- ❑ Focuses specifically on **software**.
- ❑ **Key Message:** Requirements Engineering is recognized as a critical, standalone knowledge area within the software engineering profession.

Real-World Challenges (1/2)

Challenge 1: The "I'll Know It When I See It" Mentality

- Stakeholders have a vague idea but struggle to articulate it.
- **Solution:** Use prototypes, mockups, and examples to make requirements visual.

Challenge 2: Unclear or Conflicting Needs

- Different stakeholders want different, often contradictory, things.
- Marketing vs. Engineering vs. Legal.

Solution: Facilitated workshops and negotiation.

Real-World Challenges (2/2)

Challenge 3: Requirements Volatility

- **The only constant is change.** Market conditions, technology, and understanding evolve.
- **Solution:** Have a clear change management process. Embrace iterative development.

Challenge 4: The Communication Gap

- **"The door problem":** A user, a project manager, and a developer will all describe a "door" differently.
- **Solution:** Active listening, clear documentation, and a shared vocabulary.

03

The RE Toolkit
Elicitation and Documentation Techniques
RE 工具包
引出和记录技术



What is Elicitation?

Elicitation: The Art of Asking the Right Questions

Elicitation is the practice of researching and discovering the requirements of a system from users, customers, and other stakeholders. It's not just about writing down what people say; it's about **uncovering what they truly need**.

Core Elicitation Technique

1. Interviews

- ❑ **What:** One-on-one or small group conversations.
- ❑ **Formal:** Scheduled, with a prepared list of questions.
- ❑ **Informal:** Ad-hoc, conversational.
- ❑ **Best For:** Gaining deep, qualitative insights; building rapport.
- ❑ **Pros:** Rich data, flexible, clarifies misunderstandings.
- ❑ **Cons:** Time-consuming, difficult to analyze, subject to interviewer bias.

Core Elicitation Technique

2: Workshops (e.g., JAD Sessions)

- ❑ **What:** Structured, collaborative meetings with key stakeholders present.
- ❑ **JAD = Joint Application Design**
- ❑ **Best For:** Resolving conflicts, building consensus, generating ideas quickly.
- ❑ **Pros:** Fast decision-making, shared understanding, high engagement.
- ❑ **Cons:** Can be dominated by vocal individuals, requires a skilled facilitator.

Core Elicitation Technique

3: Observation / Ethnography

- ❑ **What:** "Shadowing" users in their natural environment to see how they *actually* work.
- ❑ **Best For:** Understanding unarticulated or complex workflows; identifying inefficiencies.
- ❑ **Pros:** Reveals real-world processes that users might not think to mention.
- ❑ **Cons:** Time-intensive, can make users nervous ("Hawthorne Effect").

Core Elicitation Technique

4: Surveys & Document Analysis

Questionnaires & Surveys

- ❑ **What:** Written set of questions for a large audience.
- ❑ **Best For:** Gathering quantitative data from a large, geographically dispersed group.
- ❑ **Pros:** Reaches many people, easy to analyze statistically.
- ❑ **Cons:** Low response rates, no opportunity for follow-up questions.

Document Analysis

- ❑ **What:** Studying existing systems, procedures, forms, and documentation.
- ❑ **Best For:** Understanding current processes, identifying data sources.
- ❑ **Pros:** Provides concrete facts, not opinions.
- ❑ **Cons:** Documents may be outdated; shows "as-is," not "to-be."

Core Elicitation Technique 5: Brainstorming

- ❑ **What:** Group activity for generating a large number of creative ideas without criticism.
- ❑ **Best For:** Innovation, exploring new features, solving problems.
- ❑ **Pros:** Encourages creativity, involves the whole team.
- ❑ **Cons:** Requires strict facilitation to avoid early criticism; ideas need to be refined later.

The Key Principle of Elicitation (The Golden Rule): No Single Technique is Enough

- ❑ You will almost always use a **combination of techniques** to get a complete picture.
- ❑ **Example:** Use a survey to identify common issues, then conduct interviews to explore them in depth, and finally hold a workshop to prioritize solutions.

Once We Have the Info, How Do We Write It Down? From Elicitation to Documentation

Elicitation gives us the raw material.

Documentation structures it so everyone understands it the same way. There is no single "right" way to document—it depends on the project!

Documentation Style 1:

Traditional: Software Requirements Specification (SRS)

- ❑ A comprehensive, text-heavy formal document.
- ❑ **Structure:** Includes Introduction, Overall Description, Specific Requirements (functional, non-functional), Appendices.
- ❑ **Best For:** Safety-critical systems (medical, aviation), large government contracts, projects where formal sign-off is required.
- ❑ **Pros:** Very detailed, unambiguous, legally robust.
- ❑ **Cons:** Can be slow to produce and update; can be difficult to read.

Documentation Style 2: Agile (User Stories)

- ❑ **Format:** "As a **[role]**, I want to **[action]**, so that **[benefit]**."
- ❑ **Example:** "As a **buyer**, I want to **filter products by price**, so that **I can stay within my budget**."
- ❑ **Focus:** The *conversation* about the requirement is more important than the written text.
- ❑ **Best For:** Agile/Scrum projects, fast-changing environments.
- ❑ **Pros:** Simple, user-centric, easy to prioritize.

Documentation Style 3: Use Cases

Describe the interaction between a user (an actor) and the system to achieve a specific goal.

Includes: Main success path, alternate paths, and exception paths.

Visual Aid: Use Case Diagrams show actors and use cases at a glance.

Example Use Case Diagram:

[Customer] ---> (Place Order)

[Customer] ---> (Track Order)

[Admin] -----> (Manage Inventory)

Best For: Capturing complex user-system interactions.

Documentation Style 4: Models & Diagrams (Visual Documentation)

- ☐ A picture is worth a thousand words.
- ☐ **Wireframes/Mockups:** Show the layout and UI.
- ☐ **Flowcharts/Activity Diagrams:** Show the steps in a process.
- ☐ **Entity-Relationship Diagrams (ERD):** Show data relationships.
- ☐ **Prototypes:** Clickable models that simulate the final product.
- ☐ **Best For:** Communicating with stakeholders, validating understanding early.

Matching Documentation to Project Context How to Choose?

- ❑ **Building an Air Traffic Control System? -> SRS** (need for formality and precision).
- ❑ **Building a new social media feature? -> User Stories & Prototypes** (need for speed and user feedback).
- ❑ **Replacing a complex business process? -> Use Cases & Flowcharts** (need to understand workflow).

The best method depends on the project's needs.

Session 3 Recap What We Covered

- **Elicitation Techniques:** Interviews, Workshops, Observation, Surveys, Document Analysis, Brainstorming.
- **Key Idea:** Use a combination of techniques.
- **Documentation Styles:** Traditional (SRS), Agile (User Stories), Use Cases, and Models/Diagrams.
- **Key Idea:** The right documentation style depends on the project context.

04

Ensuring Success
Validation, Verification, and Management

确保成功
验证、验证和管理



The Final Hurdles: V&V and Management The Two Final Challenges

1.Quality: Are our requirements correct and complete?

2.Change: What happens when requirements inevitably evolve?

Validation & Verification (V&V) - A Critical Distinction Validation vs. Verification

- Two of the most important concepts in RE.
- **Validation:** "Are we building the **right** thing?" Checks that the requirements accurately reflect stakeholder needs.
 - **Focus:** *Correctness and completeness of the **requirements**.*
- **Verification:** "Are we building the thing **right**?" Checks that the finished software correctly implements the requirements.
 - **Focus:** *Correctness of the **implementation** (code).*
 - This is primarily achieved through **testing**.

V&V:

Building a House Analogy

- **Validation:** Reviewing the blueprint with the home owner.
- **Question:** *"Does this blueprint reflect your dream house?"*
- This is a **requirements-level** check.

- **Verification:** The building inspector checking the constructed house against the blueprint.
- **Question:** *"Was the house built according to the blueprint and code?"*
- This is a **testing-level** check.

You must do both!

Validation Techniques: "Are we building the right thing?" How to Validate Requirements

- ❑ **Reviews & Walkthroughs:** Formal meetings where stakeholders examine the requirements document line-by-line.
- ❑ **Prototyping:** Creating a mock-up or simple version for users to interact with and provide feedback. *"Is this what you meant?"*
- ❑ **User Acceptance Criteria (UAC):** Defining specific, testable conditions that must be met for the user to accept a feature.

Change is Inevitable: Requirements Management The Only Constant is Change
Market conditions, technology, and stakeholder understanding will evolve.

Requirements Management is the process of managing these changes in a controlled way, rather than letting them cause chaos.

Change Management Process

A formal process to evaluate and incorporate changes.

- 1. Change Request:** Anyone can submit a request for a change.
- 2. Impact Analysis:** The team analyzes the effect on budget, schedule, and other requirements.
- 3. Decision (CCB):** A Change Control Board (or key stakeholders) approves or rejects based on the analysis.
- 4. Implementation:** If approved, the change is scheduled, and **all documentation is updated.**

This prevents "scope creep" – the uncontrolled expansion of project scope.

Requirements Traceability

The ability to link requirements forward and backward.

- ❑ **Backward (Source):** Where did this requirement come from? (e.g., a specific business goal, a user interview).
- ❑ **Forward (Implementation):** Where is this requirement implemented? (e.g., specific design element, code module, test case).
- ❑ **Why is this SO important? Impact Analysis:** If a requirement changes, traceability tells you what code and tests are affected.
- ❑ **Coverage:** Ensures all requirements are implemented and tested.
- ❑ **Compliance:** Essential for regulated industries (e.g., proving a safety requirement was met).

Traceability Matrix (Example)

This matrix provides a clear map of the requirements throughout the lifecycle.

Requirement ID	Requirement Description	Source (User Story ID)	Design Element	Test Case ID
FR-101	User can reset password	US-15	PasswordReset.ctl	TC-204
NFR-205	Page load < 2 sec	BR-01 (Increase Sales)	CDN Configuration	TC-511 (Perf Test)
...

Prioritization: You Can't Do Everything at Once

Requirements Prioritization

Time and resources are limited. We need to decide what is most important.

A Common Technique: MoSCoW Method

- ☐ **M - Must have:** Non-negotiable. The project fails without it.
- ☐ **S - Should have:** Important but not vital. Could be postponed.
- ☐ **C - Could have:** Desirable but less important. "Nice to have."
- ☐ **W - Won't have** (this time): Agreed to be excluded from the current scope.

This helps manage stakeholder expectations and deliver the highest value first.

The Core Message Reiterated

Requirements Engineering is a Foundational Skill

It is not about creating paperwork. It is about:

- ❑ **Critical Thinking:** Analyzing problems deeply.
- ❑ **Communication:** Bridging the gap between people and technology.
- ❑ **Negotiation:** Balancing conflicting needs.
- ❑ **Forethought:** Anticipating change and risk.

Mastering RE is one of the highest-leverage skills for a successful career in software.

Final Thoughts & The Path Forward

- ❑ Requirements Engineering is a deep and rewarding discipline.
- ❑ Practice these concepts in your projects—start by identifying stakeholders and writing clear user stories.
- ❑ The best way to learn is by doing.

Thank you for your attention and engagement throughout this course!