Table of Contents

Chapter 1

Briefing Document: Fundamentals of Software Requirements Engineering

Executive Summary

Software Requirements Engineering (RE) is the foundational and most critical phase of the software development lifecycle, focused on discovering, defining, and managing the purpose of a software system. Its successful execution is the primary determinant of project success, as errors introduced during this stage are both the most common and the most costly to rectify. Data indicates that requirements-related issues account for 56% of all defects in software products and consume an astounding 82% of the total effort required to fix defects.

High-profile failures, such as the $235.9 million Mars Climate Orbiter loss due to a simple unit mismatch and the cancellation of the $400 million GIRES project from an inability to manage changing requirements, serve as stark evidence of the catastrophic consequences of inadequate RE. The process is not a single activity but an iterative cycle of core disciplines: **Elicitation** (gathering needs), **Analysis** (understanding needs), **Specification** (documenting needs), and **Validation** (confirming needs), all under the umbrella of continuous **Requirements Management**.

A central tenet of RE is navigating the complex matrix of needs from diverse stakeholders, ranging from end-users to senior management. This involves distinguishing between different levels of requirements—including **Business**, **User**, **Functional**, and **Non-functional**—to build a comprehensive and unambiguous specification that guides the entire development effort. A disciplined approach to RE is therefore not an optional formality but an essential risk mitigation strategy for delivering valuable and correct software solutions.

1. The Critical Importance of Requirements Engineering

The ultimate success of a software system is measured by the extent to which it satisfies its intended purpose. Requirements Engineering is the systematic process of discovering, defining, and documenting that purpose. It is considered the most challenging conceptual phase of software development, with profound implications for the project's outcome.

The Foundational Challenge

Defining software requirements is cited as "the hardest single part of conceptually defining a software system." The significance of this phase was articulated by Frederick Brooks in 1987, stating, **"no other part of the work so cripples the resulting system if done wrong"** and **"no other part is more difficult to rectify later."** This underscores that errors made in defining what a system should do are far more damaging and difficult to repair than errors made in implementation.

The Cost of Failure: Statistical Evidence

The economic and practical impact of poor requirements practices is immense. The worldwide software industry operates on a scale of billions of dollars annually, and a significant portion of project costs is dedicated to rework caused by defects originating in the requirements phase.

Defect Analysis	Percentage
Distribution of Defects by Origin	
Requirements	56%
Design	27%
Other	10%
Code	7%
Distribution of Effort to Fix Defects	
Requirements	82%
Design	13%
Other	4%
Code	1%

These figures demonstrate a critical imbalance: while requirements activities are the source of over half of all defects, fixing them consumes over four-fifths of the total corrective effort, highlighting the immense value of investing in quality RE upfront.

Case Studies in RE Failure

Mars Climate Orbiter (1999)

- **Mission:** A NASA project to study the Martian climate.
- **Cost:** Approximately $235.9 million.
- **Failure:** The orbiter was lost as it approached Mars. The root cause was a software requirements failure where two systems—one from NASA using the metric system and one from contractor Lockheed Martin using the English system—could not correctly interface due to the measurement mismatch. This was a fundamental failure in specifying an external interface requirement.

Integrated Resource Management Project (GIRES)

- **Mission:** A project to replace over 1,000 legacy systems across 140 organizations in Canada.
- **Scope:** The project, initiated in 1998, had an estimated budget of $80 million and an 8-year timeline.
- **Failure:** The project was unable to cope with changing requirements and was ultimately canceled in 2003 after a total expenditure of $400 million. This illustrates the critical need for robust requirements management processes.

2. Core Concepts and Terminology

A precise understanding of key terms and concepts is essential for effective Requirements Engineering. This includes defining what a requirement is, identifying its sources, and classifying its different types.

Definition of a Software Requirement

A software requirement has two complementary definitions:

1. **IEEE/ISO/IEC 29148-2018:** ”A statement which translates or expresses a need and its associated constraints and conditions.”
2. **SWEBOK V3:** ”At its most basic, a software requirement is a property that must be exhibited by something in order to solve some problem in the real world.”

Collectively, the set of requirements for a system forms a complex matrix representing the needs of numerous stakeholders, including end-users, domain experts, support staff, auditors, senior management, and development team members.

Categorization of Requirements

Requirements are categorized into a hierarchy that moves from high-level business objectives to detailed system specifications.

Requirement Type	Definition	Example
Business	Describes the high-level goals of the organization for undertaking the project.	"The airline wants to reduce airport counter staff costs by 25 percent."
User	Describes the goals or tasks that users must be able to perform with the system.	"A user needs to be able to check-in for their flight using the airline's website."
Functional	Specifies the behaviors the product will exhibit under specific conditions.	"If the Passenger's profile does not indicate a seating preference, the reservation system shall assign a seat."
Non-functional	Describes how well the product performs its functions (quality attributes, interfaces, constraints).	"The airline's website needs to be available 98% of the time."

Functional vs. Non-functional Requirements

The distinction between what a system *does* and *how well* it does it is a central concept in RE.

Aspect	Functional Requirements	Non-Functional Requirements (NFRs)
Focus	Product features ("what" the system does)	Product properties ("how" the system does it)
Purpose	Helps verify functionality	Helps verify performance and quality
Capture	Generally easier to define and capture	Often more difficult to capture precisely
Example	"When a site visitor creates an account, the server shall send a welcome email."	"When sending welcome emails, the server must send them within 10 minutes of registration."

Deconstructing Non-Functional Requirements (NFRs)

NFRs can be further broken down into three main categories:

NFR Sub-Type	Definition	Example
Quality Attributes	Describes the product's characteristics (performance, safety, usability, availability, etc.).	"The flight search results should load within 5 seconds after the user hits the 'Search' button."
External Interfaces	Describes connections between the system and the outside world (other software, hardware, users).	"A link to the Tawakkalna system database needs to be established to verify the COVID-19 status of a passenger."
Constraints	Imposes restrictions on the options available to the developer (process, programming language, standards).	"The system development process and deliverable documents shall conform to the process and deliverables defined in MIL-STD-498."

For NFRs to be effective, they must be specific and measurable. Vague statements should be refined into quantifiable metrics:

- **Qualitative:** "The system should allow multiple users to login concurrently."
- **Quantitative with Measure:** "The system should allow 10,000 users to login concurrently."
- **Quantitative with Measure & Metric:** "The system should allow 10,000 users to login concurrently every minute."

Domain Requirements

These are requirements derived from the specific application domain of the system. They often act as constraints on existing requirements or introduce new functionalities that are specific to that field.

- **Example:** "A patient's designated physician is the only entity allowed to retrieve a patient's medical reports."

3. The Requirements Engineering Process

RE is an iterative, multi-faceted process, not a discrete, one-time activity. It encompasses two major sub-disciplines: Requirements Development and Requirements Management.

Requirements Development (RD)

RD is the process of defining the system's requirements and involves a cycle of "progressive refinement of detail" across four key activities:

1. **Elicitation:** Discovering and gathering requirements from stakeholders through techniques like interviews and document analysis.
2. **Analysis:** Developing a deep understanding of the requirements, identifying conflicts and gaps, and decomposing high-level needs into details.
3. **Specification:** Documenting the requirements in a clear, well-organized, and unambiguous format suitable for various audiences (e.g., in a Software Requirements Specification document).
4. **Validation:** Reviewing the documented requirements with stakeholders to confirm completeness, correctness, and alignment with their needs.

These activities are highly iterative, with constant feedback loops. For example, validation may lead to rewriting the specification, analysis may reveal gaps that require further elicitation, and so on.

Requirements Management (RM)

RM begins after a set of requirements has been agreed upon and baselined. Its purpose is to handle the inevitable changes to requirements throughout the project's lifecycle. Key RM activities include:

- Tracing requirements to their origins and to subsequent design and test artifacts.
- Tracking the status of requirements.
- Managing changes through a formal process that assesses the impact of proposed changes before accepting or rejecting them.

The boundary between RD and RM is the creation of **Baselined Requirements**. RD focuses on establishing this baseline, while RM focuses on maintaining its integrity over time.

4. Common Challenges in Requirements Engineering

The RE process is fraught with potential pitfalls that can undermine a project. The famous "tree swing" cartoon, which illustrates the vast gulf between a customer's needs and the final product due to misinterpretation at every stage, serves as a powerful metaphor for these challenges.

Key challenges include:

- **Stakeholder Issues:** Insufficient user involvement or overlooking key stakeholders can lead to a system that fails to meet critical needs.
- **Lack of Expertise:** The team may lack the necessary domain or technical expertise to correctly define requirements.
- **Ambiguity:** Requirements that are unclear, ambiguous, or conflicting can lead to incorrect implementation.
- **Unrealistic Expectations:** Unrealistic or unclear project goals can set the project up for failure from the start.
- **Change Management:** Uncontrolled changes, known as "creeping user requirements" or scope creep, can derail timelines and budgets. All requirements are subject to change, and managing this is a core challenge.

Glossary of Key Terms

Term	Definition
Analysis	An RE activity to develop a deep understanding of the requirements, involving tasks like decomposition and identifying conflicts or gaps.
Business Requirements	Describe why the organization is implementing the system.
Constraints	A type of non-functional requirement that imposes restrictions on the options available to the developer during construction (e.g., programming language, cost, delivery date).
Domain Requirements	Requirements derived from the application domain that describe system characteristics reflecting that domain. They can be new functional requirements or constraints.
Elicitation	An RE activity that involves all tasks associated with discovering and gathering requirements from stakeholders (e.g., interviews, document analysis).
External Interfaces	A type of non-functional requirement that describes the connections between the system and the outside world (e.g., other software, hardware, users).
Functional Requirements	Specify the behaviors the product will exhibit under specific conditions. They describe *what* the system should do.
Non-functional Requirements	Describe *how well* the product carries out its tasks. They define product properties and include quality attributes, external interfaces, and constraints.
Product Owner	A key stakeholder who serves as the communication link between the development team and other external stakeholders.
Quality Attributes	A type of non-functional requirement describing the product's characteristics in various dimensions important to users or developers (e.g., performance, safety, usability, availability). Also called quality of service requirements.
Requirements Development (RD)	The sub-process of RE that includes the activities of elicitation, analysis, specification, and validation.

Requirements Engineering (RE)	The process of discovering a system's purpose by defining a set of software requirements to delineate the constraints and demands of the system.
Requirements Management (RM)	The RE activity of managing requirements after they have been agreed upon, including tracing requirements, tracking status, and managing changes.
Software Requirement	A property that must be exhibited by something to solve a real-world problem (SWEBOK V3). It is also defined as a statement that translates or expresses a need and its associated constraints and conditions (IEEE/ISO/IEC 29148-2018).
Software Requirements Specification (SRS)	The document that is the final output of the requirements development process, containing the complete set of functional and non-functional requirements for a system.
Specification	An RE activity focused on documenting requirements in a well-organized way suitable for comprehension by their intended audiences.
Stakeholders	Individuals or groups spread across different levels of an organization who are somehow linked to the problem the software aims to solve.
System Requirements	Describe the requirements for a product that is composed of multiple components or subsystems.
User Requirements	Describe the goals or tasks that users must be able to perform with the product.
Validation	An RE activity where documented requirements are reviewed by internal teams and the customer to confirm completeness, proper representation, and that they meet customer needs.

Chapter 2

Requirements Inception: A Synthesis

Executive Summary

Requirements Inception is a critical initial phase in project development, designed to establish a unified and clear understanding of a product's context. Its primary goal is to define the underlying problem, the proposed solution, its boundaries, and its benefits, thereby preventing project disasters such as unaligned objectives, stakeholder disagreements, missed deadlines, and budget overruns. The principal output of this phase is the **Vision and Scope Document**, a foundational text that guides the entire project. This document is structured around three core pillars: **Business Requirements**, which detail the primary benefits and objectives; **Scope and Limitations**, which define what the product will and will not do; and **Business Context**, which outlines stakeholder profiles, project priorities, and deployment considerations. To ensure clear communication of the product's boundaries, various scope representation techniques are employed, including Context Diagrams, Ecosystem Maps, and Feature Trees. The guiding philosophy of this phase is to conduct just enough investigation to form a rational and justifiable opinion on the project's purpose and feasibility before committing to deeper exploration.

1. The Rationale for Requirements Inception

The inception phase serves as the strategic foundation for any new system or product. Its objective is to create a shared understanding among all stakeholders regarding the product's context, ensuring alignment on the problem being solved, the nature of the solution, the system's boundaries, and the value it will deliver.

The Imperative for a Clear Direction

Embarking on a project without a well-defined and clearly communicated direction is a direct invitation to failure. The primary risks associated with neglecting the inception phase include:

- **Unaligned Objectives:** Different teams and stakeholders work towards conflicting or divergent goals.
- **Stakeholder Disagreement:** Lack of consensus on requirements leads to continuous conflict and revision.
- **Project Delays and Overruns:** Ambiguity and rework inevitably result in missed deadlines and exceeded budgets.

As articulated by Craig Larman, the principle of inception is to perform a focused, preliminary investigation: *"The idea is to do just enough investigation to form a rational, justifiable opinion of the overall purpose and feasibility of the potential new system, and decide if it is worthwhile to invest in deeper exploration."*

2. The Requirements Hierarchy and Documentation

The inception phase initiates a structured flow of requirements development, where high-level business goals are progressively refined into detailed technical specifications. A clear hierarchy ensures traceability from top-level objectives down to individual features.

The process follows a logical cascade:

1. **Business Requirements** articulate the high-level goals and benefits. These are captured in the **Vision and Scope Document**.
2. The Vision and Scope Document guides the definition of **User Requirements**, which describe the tasks users need to accomplish. These are documented in the **User Requirements Document**.
3. User Requirements are then translated into **System Requirements** and **Functional Requirements**, which specify the system's behavior and capabilities. These form the core of the **Software Requirements Specification (SRS)**.

This entire process is influenced by several key factors:

- **Business Rules:** Policies and constraints that govern business operations.
- **Quality Attributes:** Non-functional requirements like performance, security, and usability.
- **External Interfaces:** Connections to other systems or hardware.
- **Constraints:** Technical or business limitations imposed on the project.

3. The Vision and Scope Document: A Comprehensive Overview

The Vision and Scope Document is the primary deliverable of the requirements inception phase. It summarizes the rationale, context, and boundaries of the new product, ensuring all stakeholders are aligned. The document is logically divided into three main sections.

3.1. Business Requirements

This section describes the primary benefits the new system will provide to its sponsors, buyers, and users. These requirements directly influence the prioritization and sequencing of user requirements. They are typically sourced from funding sponsors, marketing managers, corporate executives, and product visionaries.

Key Components of Business Requirements

- **Business Problems and Objectives:** Problems describe current obstacles preventing the business from reaching its goals, while objectives define measurable ways to achieve those goals. The two are intertwined; understanding one helps reveal the other. The process is iterative, involving asking probing questions ("What is keeping us from reaching this goal?", "Why do we care about that goal?") to move from high-level problems to concrete objectives and, ultimately, to a list of features.
- **Success Metrics:** These are indicators used to define and measure success, both within and outside the organization. They confirm whether a project is on track to meet its business objectives. Metrics can be financial or non-financial.

Financial Success Metrics	Nonfinancial Success Metrics
Capture a market share of X% within Y months.	Achieve a customer satisfaction measure of at least X within Y months of release.
Increase market share in country W from X% to Y% within Z months.	Increase transaction-processing productivity by X% and reduce data error rate to no more than Y%.
Reach a sales volume of X units or revenue of $Y within Z months.	Develop an extensible platform for a family of related products.
Achieve X% return on investment within Y months.	Develop specific core technology competencies.

- **Vision Statement:** A concise summary of the project's long-term purpose. It should be balanced to satisfy diverse stakeholders and grounded in reality, even if idealistic. A common template is:
    - **For** [target customer]
    - **Who** [statement of need or opportunity]
    - **The** [product name]
    - **Is a** [product category]
    - **That** [key benefit, compelling reason to buy]
    - **Unlike** [primary competitive alternative]
    - **Our product** [statement of primary differentiation]
- **Business Risks:** Summarizes major risks involved with developing—or not developing—the project, including potential loss, likelihood, and mitigation actions.
- **Business Assumptions and Dependencies:** Documents all assumptions made by stakeholders and identifies major dependencies on external factors.

3.2. Scope and Limitations

This section clearly defines what the product is and, just as importantly, what it is not. This helps manage stakeholder expectations and prevent "scope creep"—the uncontrolled growth of project scope.

Product Vision vs. Project Scope

- **Product Vision:** Succinctly describes the ultimate product that will achieve the business objectives. It applies to the product as a whole.
- **Project Scope:** Identifies the portion of the ultimate vision that a specific project or release will address. It is more dynamic and pertains to a particular iteration.

A single product vision can encompass the scope for multiple releases, with the scope for future releases being less defined than for the current one.

Defining Scope at Multiple Levels

- **Highest Level:** Scope is defined by the business objectives the project will target.
- **Lower Level:** Scope is defined in terms of the features, use cases, or events to be included.
- **Lowest Level:** Scope is defined by the specific functional requirements planned for a release.

Key Components of Scope and Limitations

- **Major Features:** A high-level list of distinguishing product features.
- **Scope of Initial Release:** The features planned for the first project iteration, focusing on those that deliver the most value earliest.
- **Scope of Subsequent Releases:** A release roadmap outlining features for future versions.
- **Limitations and Exclusions:** A list of features or capabilities that stakeholders might expect but are explicitly not planned for inclusion.

3.3. Business Context

This section provides a summary of the project's environment, including its stakeholders and operational considerations.

Key Components of Business Context

- **Stakeholder Profiles:** Identifies all people or groups involved in the project, affected by its outcome, or able to influence it. Each profile should include their major value from the product, anticipated attitude, features of interest, and any constraints they impose.
- **Project Priorities:** An agreed-upon set of priorities that guides decision-making when changes or conflicts arise.
- **Deployment Considerations:** Summarizes information needed for effective product deployment, such as network access, data storage requirements, and data migration plans.

4. Techniques for Scope Representation

To foster clear and accurate communication of scope, it is critical to use standard, visual notation techniques.

- **Context Diagrams:** A visual illustration of the boundary between the system and the rest of the universe. It identifies external entities (terminators) that interface with the system and the data, control, or material flows between them. It does not provide visibility into the system's internal workings.
    - *Example:* A context diagram for a Chemical Tracking System would show the central system interacting with external entities like a Chemist, Buyer, Chemical Stockroom, Bar Code Reader, and Health and Safety Department via data flows such as `vendor catalog query`, `vendor order status`, and `chemical usage report`.

- **Ecosystem Maps:** Shows all systems related to the system under development, including both direct and indirect interactions. This provides a broader view than a context diagram, which only shows entities that directly interface with the system.
    - *Example:* An ecosystem map for the Chemical Tracking System might show its connections to a Purchasing System, Receiving System, Corporate Training Database, and an OSHA/EPA Reporting Interface.
- **Feature Trees:** A visual, hierarchical depiction of a product's features, organized into logical groups. Features can be subdivided into multiple levels (L1, L2, L3) to show increasing detail. The scope of a release can be defined by selecting a specific set of features from the tree.
- **Event Lists:** An inventory of external events that can trigger a response from the system. Events can be initiated by a user, triggered by time, or be a signal from an external component. The scope of a release can be defined in terms of the events the system will handle.
    - *Example Events for a Chemical Tracking System:*
        - Chemist places a chemical request. (User-triggered)
        - Chemical container bar code is scanned. (User-triggered/HW device)
        - Time to generate OSHA compliance report arrives. (Time-triggered)
        - Vendor indicates chemical is backordered. (Signal from external component)

5. Conclusion

The core purpose of the requirements inception phase can be distilled into a single, actionable directive: **Agree on a well-defined project's vision, scope, and business case.** By systematically addressing business requirements, scope, and context, teams build the necessary alignment and clarity to navigate the complexities of product development successfully.

Glossary of Key Terms

Term	Definition
Business Context	The section of the Vision and Scope Document that outlines stakeholder profiles, project priorities, and deployment considerations.
Business Objectives	Statements that define ways to measure the achievement of business goals. They address the problems that keep the business from meeting its goals.
Business Requirements	A description of the primary benefits that a new system will provide to its sponsors, buyers, and users. These requirements directly influence which user requirements are implemented and in what order.
Context Diagram	A scope representation technique that visually illustrates the boundary between the system being developed and its environment. It identifies external entities that interact with the system and the data flows between them.
Ecosystem Map	A scope representation technique that shows all systems related to the system being developed (whether directly or indirectly) and the interactions among them.
Event List	A scope representation technique that identifies external events that could trigger behavior in the system. The scope of a release can be defined in terms of the events the system will handle.
External Entities	Also known as terminators, these are entities outside the system that connect to it, such as user classes, organizations, other systems, or hardware devices. They are a key component of Context Diagrams.
Feature Tree	A scope representation technique that provides a visual depiction of a product's features organized in logical, hierarchical groups (e.g., L1, L2, L3).
Product Vision	A succinct description of the ultimate product that will achieve the business objectives. It applies to the product as a whole over its lifetime.

Project Scope	The portion of the product vision that a specific project or iteration will address. It is more dynamic than the product vision.
Requirements Inception	The initial phase of a project focused on establishing a shared understanding of the product's context, including its underlying problem, proposed solution, boundaries, and benefits. Its goal is to do just enough investigation to justify deeper exploration.
Scope Creep	The continuous or uncontrolled growth in a project's scope, often occurring as more and more functionality is added beyond the original agreement.
Stakeholders	People or groups who are involved in a project, affected by its outcome, or able to influence its outcome.
Success Metrics	Indicators used to define and measure success, both within and outside the organization. They can be financial (e.g., capture market share) or nonfinancial (e.g., achieve customer satisfaction) and indicate if a project is on track to meet its business objectives.
Vision and Scope Document	A document that is the primary result of the Requirements Inception phase. It summarizes the rationale and context for a new product, including business requirements, scope, limitations, and business context.
Vision Statement	A concise summary that communicates the long-term purpose of a project. It is written to be balanced, satisfying diverse stakeholders, and grounded in reality while potentially being idealistic.

Chapter 3

Briefing Document: The Requirements Elicitation Process

Executive Summary

Requirements elicitation is a foundational, incremental, and highly challenging aspect of software development, characterized as being critical, error-prone, and communication-intensive. Its primary goal is to gather comprehensive information about the project's domain, problems, stakeholder needs, and constraints to produce an initial requirements document.

The process is structured around three core phases: **Preparation**, **Execution**, and **Follow-up**. Preparation involves meticulously planning the scope and agenda, securing resources, and developing initial questions and models. The execution phase consists of the elicitation sessions themselves, where diligent note-taking is paramount. The follow-up phase focuses on organizing, sharing, and reviewing the gathered information to identify gaps and open issues.

Key sources for requirements are diverse, including existing systems and documentation, but stakeholders—such as clients, users, and domain experts—are the most critical. Successful elicitation depends on navigating significant challenges, including scope ambiguity, unstated requirements, and stakeholder-related issues like conflicting views or limited availability. Mitigation strategies involve rigorous scope management, domain research, and leveraging strong interpersonal and communication skills. The process is considered complete when new discussions cease to yield new, high-priority, in-scope functional requirements.

1. Core Concepts of Requirements Elicitation

1.1. Goals and Purpose

The primary objectives of the requirements elicitation process are to:

- Identify all potential sources of requirements and select the most appropriate elicitation techniques for each.
- Gather detailed information regarding the problem domain, the specific problem to be solved, stakeholder needs, and operational constraints.
- Produce a first draft of a requirements document, which will primarily contain user requirements and elicitation notes. This initial document is expected to be potentially incomplete, disorganized, and inconsistent, but serves as the necessary starting point.

1.2. Nature of the Process

Elicitation is described as "perhaps the most challenging, critical, error-prone, and communication-intensive aspect of software development." It is an incremental and iterative process that cycles through Elicitation, Analysis, and Specification. The aim is not merely to record stakeholder requests but to "extract the essence of stakeholders' requirements and invent new ways for them to better perform their tasks."

2. Sources of Requirements

A comprehensive elicitation process draws information from multiple sources to build a complete picture of the required system.

2.1. Primary Sources

- **Stakeholders:** Individuals or groups who are affected by or have an interest in the system.
- **Existing Systems:** Analysis of current systems, whether manual or automated, that the new system will replace or interact with.
- **Existing Documentation:** Manuals, business process descriptions, and reports related to the current system or domain.
- **Competing Systems:** Analysis of competitor products to understand market standards and potential features.

- **Interfacing Systems Documentation:** Specifications for other systems that the new product must connect with.
- **External Factors:** Standards, policies, collective agreements, and legislation that impose constraints or requirements on the system.

2.2. Key Stakeholder Roles

Stakeholder Role	Description
Client	The individual or entity that pays for the software and acts as the project sponsor.
User	Current or future operators of the system, often comprising various classes with different needs.
Domain Expert	An individual with deep familiarity with the problem domain and the environment in which the system will operate.
Developer	The technical team member responsible for assessing technical feasibility.
Others	Includes roles such as the **Project Manager** and **Tester**, who have specific interests in the project's requirements.

3. The Three-Phase Elicitation Process

The elicitation process is a structured workflow composed of distinct planning, execution, and review phases.

Phase 1: Prepare for Elicitation

Thorough preparation is critical for the success of any elicitation activity. This phase involves both high-level planning and detailed session preparation.

A. Planning for Elicitation:

- **Objectives:** Define overall goals for the elicitation effort as well as specific goals for individual activities.
- **Strategy & Techniques:** Develop a strategy that pairs appropriate elicitation techniques with specific stakeholders.
- **Schedule & Resources:** Create a schedule and allocate resources, coordinating between clients/customers and the development team.

- **Independent Elicitation:** Plan for activities that can be done independently, such as the analysis of existing documents and systems.
- **Expected Products:** Define the target deliverables, such as use cases, a Software Requirements Specification (SRS), or a quality attributes specification.
- **Risks:** Identify potential risk factors and formulate plans to overcome them.

B. Session Preparation:

- **Decide on Scope and Agenda:** The scope should be clearly defined in terms of topics, questions, process flows, or use cases. An agenda should list all topics, assign time slots, and state objectives.
- **Prepare Resources:** Assemble all necessary physical resources, confirm participants, and gather relevant documentation or system access.
- **Stakeholder Analysis:** Identify relevant stakeholders and learn about their cultural, regional, and language preferences to facilitate communication.
- **Prepare Questions and Straw Man Models:** Draft initial analysis models (e.g., use cases, process flows) to help users provide more targeted input. Prepare guiding questions to probe for details, such as:
    - *What else could…*
    - *What happens when…*
    - *Would you ever need to…*
    - *Where do you get…*
    - *Why do you (or don't you)…*
    - *Does anyone ever…*

Phase 2: Perform Elicitation Activities

This is the execution phase where the actual elicitation session takes place. The key activity during this phase is to take comprehensive and accurate notes that capture all critical information, including:

- A list of attendees.
- Decisions made during the session.

- Action items, with clear assignment of responsibility for each.
- Outstanding issues that require further clarification.
- Key points and highlights from the discussion.

Phase 3: Follow Up After Elicitation

The work does not end when the session is over. A structured follow-up ensures that the gathered information is accurate, organized, and actionable.

- **Organize and Share Notes:**
    - Review and update notes immediately after the session while the details are fresh.
    - Consolidate input if information was gathered from multiple sources or sessions.
    - Edit notes with care for clarity, but always keep the original, raw notes for reference.
    - Review the organized notes with stakeholders to confirm accuracy and understanding.
- **Document Open Issues:**
    - Identify and list any items that need to be explored further.
    - Pinpoint any gaps in the gathered information that need to be closed in subsequent sessions.

4. Classifying and Understanding Customer Input

Information gathered during elicitation can be categorized into several types, all of which are necessary for a complete specification.

Requirement Category	Description	Example from Source
Business Requirements	High-level goals of the organization.	“Save SAR X/year on electricity now wasted by insufficient units”.

User Requirements	Goals or tasks users must be able to perform.	"I need to print a mailing label for a package".
Business Rules	Corporate policies or regulations that constrain the system.	"Time-off approvals must comply with the company's HR vacation policy."
Functional Requirements	Specific system behaviors or functions.	"The user must be able to sort the project list in forward and reverse alphabetical order."
Quality Attributes	Non-functional requirements defining system quality.	"The mobile software must respond quickly to touch commands."
Constraints	Limits or restrictions on the system's design or implementation.	"Files submitted electronically cannot exceed 10 MB in size."
External Interface Req.	Requirements related to interaction with other systems.	"The mobile app should send the check image to the bank after I photograph the check I'm depositing."
Data Requirements	Rules governing data formats and values.	"The ZIP code has five digits, followed by an optional hyphen and four digits that default to 0000."
Solution Ideas	Stakeholder suggestions on how to implement a feature.	"Then I select the state where I want to send the package from a drop-down list."

5. Challenges and Mitigation Strategies

The elicitation process is fraught with potential challenges that can derail a project if not managed effectively.

Challenge Area	Specific Challenges	Mitigation Strategies
Scope	- Inadequately or incorrectly defined scope. - Difficulty maintaining focus on the scope.	- Re-check scope before delving into details. - Use explicit "in-scope" and "out-of-scope" lists.
Requirements	- Assumed or implied requirements. - Missing requirements.	- Actively ask, "What are we assuming here?" - Perform thorough requirements analysis.
Stakeholders	- Uncertainty or lack of clarity. - Limited technical knowledge. - Limited availability. - Lack of cooperation.	- Research the domain and consult with other experts. - Rely on the business analyst's interpersonal, communication, and interviewing skills to manage conflicts and extract information.

	- Conflicting requirements among different stakeholders.	
Business Analyst	- Lack of expertise. - Lack of domain knowledge.	- Proactively research the domain and consult with others.

6. Criteria for Completing Elicitation

Knowing when to conclude the elicitation phase is crucial. Key indicators that the process is nearing completion include:

- Users are unable to think of any more use cases or user stories.
- New scenarios proposed by users do not lead to any new functional requirements.
- Users begin to repeat issues that have already been covered.
- Newly suggested features or requirements are consistently out of the project's scope.
- Proposed new requirements are all of low priority.
- Developers and testers have few questions after reviewing the documented requirements.

7. Elicitation Best Practices

To maximize effectiveness, the following best practices are recommended:

- Define a clear product vision and project scope from the outset.
- Identify user classes and their specific characteristics.
- Select a "product champion" for each user class to act as a primary point of contact.
- Work with user representatives to identify user requirements.
- Hold structured elicitation interviews and facilitated workshops.
- Conduct focus groups with typical users.
- Observe users performing their jobs in their natural environment.
- Distribute questionnaires to gather information from a broad audience.
- Perform document analysis on existing materials.
- Examine the problems and shortcomings of the current system.

- Identify system events and the required responses to them.
- Reuse existing requirements from previous projects where applicable.

Glossary of Key Terms

Term	Definition
Business Analyst	The professional whose challenges may include a lack of expertise or domain knowledge, and whose interpersonal, communication, and interviewing skills are key mitigation factors.
Business Requirements	A category of customer input. An example is: "Save SAR X/year on electricity now wasted by insufficient units."
Business Rules	A category of customer input that dictates policy compliance. An example is: "Time-off approvals must comply with the company's HR vacation policy."
Client	A type of stakeholder who pays for the software and acts as the project sponsor.
Constraints	A category of customer input that defines limitations. An example is: "Files submitted electronically cannot exceed 10 MB in size."
Data Requirements	A category of customer input defining data formats. An example is: "The ZIP code has five digits, followed by an optional hyphen and four digits that default to 0000."
Developer	A type of stakeholder who provides input on the technical feasibility of requirements.
Domain Expert	A type of stakeholder who is familiar with the problem being solved and the environment the system will operate in.
Elicitation	The process of gathering requirements. It is described as challenging, critical, error-prone, communication-intensive, and incremental. It is part of a cycle with Analysis and Specification.
Elicitation Goals	The objectives of the process, which include determining sources of requirements, eliciting information (domain, problem, needs, constraints), and producing a first document.
External Interface Requirements	A category of customer input related to how the system interacts with other systems.
Functional Requirements	A category of customer input describing specific system behaviors. An example is: "The user must be able to sort the project list in forward and reverse alphabetical order."

Project Manager	A type of stakeholder involved in the elicitation process.
Quality Attributes	A category of customer input that describes non-functional characteristics. An example is: “The mobile software must respond quickly to touch commands.”
Solution Ideas	A category of customer input where stakeholders suggest potential implementations. An example is: “Then I select the state where I want to send the package from a drop-down list.”
Stakeholders	Individuals or groups who are a primary source of requirements, including clients, users, domain experts, developers, project managers, and testers.
Straw Man Models	Draft analysis models (e.g., use cases, process flows) prepared before an elicitation session to help users provide better input.
Tester	A type of stakeholder involved in the elicitation process.
User	A type of stakeholder who is a current or future operator of the software. Users can belong to various classes.
User Requirements	A category of customer input describing a goal or task a user needs to accomplish. An example is: “I need to print a mailing label for a package.”

Chapter 4

Comprehensive Briefing on Software Requirements Elicitation Techniques

Executive Summary

Requirements elicitation is a critical, collaborative, and analytical phase within the software requirements engineering (RE) process. It involves identifying the needs and constraints of various stakeholders to discover business, user, functional, and non-functional requirements. Success in elicitation is fundamentally tied to active user engagement and the strategic selection of techniques tailored to the specific project environment.

Key takeaways include:

- **Iterative Nature:** Elicitation is not a one-time event but a continuous cycle of discovery, analysis, and refinement.
- **Technique Diversity:** Methods are categorized into facilitated (direct stakeholder interaction) and independent (analyst-led) techniques.
- **User-Centric Focus:** Techniques like Use Cases shift the focus from what the system should do to what the user needs to accomplish.
- **Prototyping as a Tool:** Prototypes serve to clarify requirements, explore design alternatives, and reduce uncertainty early in the development lifecycle.
- **Security Integration:** The use of "Misuse Cases" allows for the early identification of security threats and the elicitation of safety requirements.

Overview of Requirements Elicitation

Requirements elicitation is defined as the process of drawing out or calling forth information. In software engineering, it is a discovery process aimed at identifying the latent or potential needs of stakeholders.

Primary Goals

- **Information Gathering:** Elicit details regarding the application domain, the specific problem, user needs, and system constraints.
- **Source Identification:** Determine the appropriate sources of requirements and select the most effective elicitation techniques.
- **Documentation:** Produce the initial set of user requirements and elicitation notes. While this initial documentation may be incomplete or inconsistent, it provides the necessary foundation for further development.

The Requirements Engineering (RE) Process

The RE process is divided into three main management areas: Inception Management, Requirements Development, and Requirements Management.

Requirements development is inherently iterative, following a logical progression:

1. **Elicitation:** Collecting and discovering information.
2. **Analysis:** Clarifying information and identifying gaps.
3. **Specification:** Writing down requirements.
4. **Validation:** Confirming and correcting the requirements with stakeholders.

This process is a loop; findings in the analysis or specification phases often necessitate a return to elicitation to close gaps or resolve conflicts.

Requirements Elicitation Techniques

Techniques are categorized based on the level of interaction between the Business Analyst (BA) and the stakeholders.

1. Facilitated Techniques (Stakeholder Interaction)

These techniques are primarily used to discover user and business requirements through direct communication.

- **Interviews:** Conducted one-on-one or in small groups.
    - *Strengths:* Effective for quick domain immersion; interviewees may feel more comfortable sharing thoughts privately than in large groups; easier to establish rapport.
    - *Best Practices:* Acquire background knowledge on the domain and interviewee; establish rapport; listen actively and paraphrase to ensure understanding; stay within scope.
- **Workshops:** Structured, concurrent meetings with a diverse group of stakeholders.
    - *Strengths:* Excellent for solving disagreements and handling tight schedules.
    - *Best Practices:* Enforce ground rules; fill specific team roles; use "parking lots" for out-of-scope items; keep the group small but inclusive of right stakeholders.
- **Focus Groups:** Facilitated sessions with representative users to explore attitudes and preferences.
    - *Strengths:* Generates subjective feedback valuable for commercial products.
    - *Note:* Participants typically do not have decision-making authority.

2. Independent Techniques (Analyst-Led)

Independent techniques focus on discovering functionality that users might not be aware of or cannot explicitly articulate.

- **Observations:** ”In the trenches” shadowing of specialists as they perform their work.
    - *Strengths:* Identifies high-risk tasks and facilitates validation of actual work processes versus documented ones.
- **Questionnaires:** Surveys designed to gather input from large or geographically distributed user populations.
    - *Strengths:* Inexpensive and easily administered.
    - *Best Practices:* Use consistent scales; avoid suggestive questions; ensure answer choices are mutually exclusive and exhaustive.
- **System Interface Analysis:** Examining the systems to which a new system will connect.
    - *Outcome:* Reveals functional requirements regarding data exchange and service integration.
- **User Interface Analysis:** Studying existing or similar systems to identify areas for improvement.
    - *Strengths:* Clarifies user input by relating it to real-world interface examples.
- **Document Analysis:** Reviewing existing documentation (procedures, regulations, standards).
    - *Risk:* Documents may be outdated and might not reflect day-to-day reality.

Use Cases and Misuse Cases

Use Cases

A use case describes a sequence of interactions between a system and an external actor that results in an outcome of value. This technique shifts the perspective from a product-centric view to a usage-centric view.

- **Usage Scenario:** A single instance of a use case (e.g., "Request a chemical from a vendor").
- **Representation:** Can be textual (narrative form) or visual (diagrams).
- **Benefits:** Prevents unneeded functionality and provides clear expectations for developers.

Misuse Cases

A misuse case is a negative scenario describing an undesirable goal from a business perspective or a desirable goal for a hostile agent.

- **Applications:** Security and risk analysis.
- **Process:** Identify potential "misactors" (e.g., crooks, competitors) and determine how they might harm the system (e.g., "Steal password" or "Flood system").
- **Mitigation:** Used to elicit requirements for system protection and message encryption.

Prototyping

A prototype is a preliminary implementation of a product used to take a tentative step into the "solution space."

Purpose and Types

Type	Mock-up (Requirement/Design Tool)	Proof of Concept (Construction Tool)
Throwaway	Clarify/refine requirements; identify missing functionality; explore UI.	Demonstrate technical feasibility; evaluate performance.
Evolutionary	Implement core requirements; grow based on priority; adapt to changing needs.	Implement/optimize core algorithms; tune performance; grow into the final product.

Fidelity

- **Low Fidelity:** Static, non-functional (e.g., paper sketches). It is quick and cheap but not interactive.
- **High Fidelity:** Fully functional or reactive (e.g., electronic/digital tools like Figma). It allows for precise decisions but is time-consuming and costly.

Strategic Selection Factors

Choosing the right technique depends on several variables:

1. **Nature of Information:** Whether the requirements are conscious or latent.
2. **Constraints:** Available time and budget.
3. **Stakeholder Availability:** Access to users and executives.
4. **BA Experience:** The analyst's familiarity with specific techniques.

Technique Application Mapping

Project Context	Recommended Techniques
Mass-market software	Interviews, Focus groups, Questionnaires
Internal corporate software	Interviews, Workshops, Focus groups, Observations, System interface analysis, Document analysis
Replacing existing system	Interviews, Workshops, Observations, System/User interface analysis, Document analysis

Geographically distributed	Interviews, Workshops, Questionnaires

Appendix: Sample Elicitation Questions

To ensure comprehensive coverage, analysts should address the following categories:

- **Functional:** What will the system do? Under what stimuli? What data transformations occur?
- **Design Constraints:** Where is equipment located? Are there size/power restrictions? What programming languages are required?
- **Performance:** What are the requirements for execution speed, response time, and data throughput?
- **Usability:** What training is required? How easy is it to understand? How difficult is it to misuse?
- **Security:** Is access controlled? Is data isolated? Are there precautions against vandalism?
- **Reliability/Availability:** What is the mean time between failures? How often is the system backed up?
- **Maintainability:** How easy is it to add features or port the system to a new platform?
- **Precision:** How accurate must calculations be? What is the required degree of precision?

Requirements Elicitation: A Comprehensive Study Guide

This study guide provides an exhaustive review of requirements elicitation as part of the software engineering process. It covers the definitions, goals, specific techniques, and the iterative nature of requirements development.

1. Overview of Requirements Elicitation

Definition and Purpose

Requirements elicitation is the collaborative and analytical process of identifying the needs and constraints of various stakeholders for a software system. The term "elicit" means to draw out or call forth something latent or potential, such as information or responses.

The process involves several core activities:

- **Collection:** Gathering raw data from stakeholders.
- **Discovery:** Finding hidden or unstated requirements.
- **Extraction:** Pulling specific details from existing systems or documentation.
- **Definition:** Formulating clear requirements from the gathered information.

Core Goals

The primary objectives of the elicitation phase include:

- Determining sources of requirements and selecting appropriate techniques.
- Eliciting information regarding the application domain, the specific problem, user needs, and system constraints.
- Discovering business, user, functional, and non-functional requirements.
- Producing a first document, which typically includes user requirements and elicitation notes. (Note: These initial documents may be incomplete or inconsistent).

The Requirements Engineering (RE) Process

Requirements development is an iterative cycle consisting of four main stages:

1. **Elicitation:** Drawing out needs.
2. **Analysis:** Studying the gathered information to clarify needs.
3. **Specification:** Writing the requirements down.

4. **Validation:** Confirming and correcting the requirements with stakeholders.

This process is repeated as analysis reveals conflicting or missing requirements, requiring a return to the elicitation phase.

2. Elicitation Techniques

Techniques are generally categorized into two types: **Facilitated** (interacting with stakeholders) and **Independent**(working alone).

Facilitated Techniques

These focus primarily on discovering user and business requirements through direct interaction.

Interviews

Interviews can be conducted one-on-one or in small groups.

- **Strengths:** Effective for learning the application domain quickly, establishing user involvement, and accommodating busy executives. Interviewees may feel more comfortable sharing thoughts privately than in large groups.
- **Tips for Success:** Acquire background knowledge on the domain and interviewee first. Prepare questions, establish rapport, stay in scope, and practice active listening and paraphrasing.

Workshops

Structured meetings where requirements are elicited from multiple stakeholders concurrently.

- **Strengths:** Highly effective for solving disagreements and suitable for tight schedules.
- **Tips for Success:** Establish ground rules, fill specific team roles, plan an agenda, and use “parking lots” for items to be considered later. Timeboxing and keeping the group small but inclusive are also critical.

Focus Groups

A representative group of users convened to generate ideas regarding functional and quality requirements.

- **Strengths:** Valuable for developing commercial products; generates subjective feedback on attitudes, impressions, and preferences.
- **Management:** Participants usually lack decision-making authority. Facilitators must keep them on topic without influencing their opinions.

Independent Techniques

These focus on discovering functionality that users might not be aware of by analyzing systems or data.

Observations

The practice of shadowing specialists "in the wild" as they perform their tasks.

- **Strengths:** Useful for important or high-risk tasks and identifies new topics for interviews.
- **Nature:** Can be silent or interactive; often time-consuming but reveals insights other techniques cannot.

Questionnaires

Surveys used to gather needs from large or geographically distributed user populations.

- **Tips for Success:** Use consistent scales, avoid suggestive questions, and ensure answer choices are mutually exclusive and exhaustive. Closed questions are preferred for statistical analysis.

System and User Interface Analysis

- **System Interface Analysis:** Examines the systems to which the new system will connect, revealing requirements for data exchange and service sharing.

- **User Interface Analysis:** Studying existing or similar products to understand current workflows and identify areas for improvement.

Document Analysis

Examining existing documentation (procedures, regulations, standards) to learn about a new domain or system. While useful, analysts must be wary as documentation is often outdated.

3. Use Cases and Misuse Cases

Use Cases and Scenarios

A **Usage Scenario** is a description of a single instance of system usage involving a specific actor, time, and data. A **Use Case** is a collection of these related scenarios.

- **Focus:** Usage-centric rather than product-centric. It discusses what users need to *accomplish* rather than just what the system should *do*.
- **Representation:** Can be textual (narrative paragraphs) or visual (diagrams with actors and ovals representing tasks).

Misuse Cases

A **Misuse Case** captures scenarios with goals that are undesirable from a business perspective or desirable to a hostile agent (a "misuser").

- **Purpose:** Essential for security and risk analysis.
- **Process:** Identify potential misactors, determine how they might harm the system (e.g., stealing passwords, flooding the system), and define mitigations.

4. Software Prototyping

A prototype is a partial or preliminary implementation used to take a tentative step into the solution space.

Types and Purposes

Type	Purpose
Mock-up	Clarify/refine user requirements and explore UI approaches.
Proof of Concept	Demonstrate technical feasibility and evaluate performance.
Throwaway	Created quickly and cheaply to be discarded after requirements are validated.
Evolutionary	A subset of functionality that grows into the final product.

Fidelity

- **Low Fidelity:** Static or non-functional (e.g., paper sketches). Easy and cheap but not interactive.
- **High Fidelity:** Fully functional and reactive. Provides deeper understanding but is costly and time-consuming.

Risks

Prototyping carries risks such as pressure to release the prototype as the final product, distraction by minor details, and investing excessive effort into something meant to be temporary.

5. Factors for Selecting Techniques

When choosing an elicitation technique, a Business Analyst (BA) should consider:

- **Consciousness:** The nature of the information (latent vs. obvious).
- **Constraints:** Available time and budget.
- **Availability:** Access to specific stakeholders.
- **Experience:** The BA's familiarity with the chosen technique.

Glossary of Key Terms

- **Actor:** An external entity (user or system) that interacts with the software to achieve a goal.
- **Analysis:** The phase of studying elicited information to clarify needs and close gaps in understanding.
- **Elicitation:** The collaborative process of collecting, discovering, and defining requirements from stakeholders.
- **Fidelity:** The extent to which a prototype is "real," reactive, and functional.
- **Misuser:** A hostile agent with intentions to harm a system, its stakeholders, or their resources.
- **Parking Lot:** A technique used in workshops to capture and store items or discussions for later consideration to keep the current session on track.
- **Prototype:** A partial or preliminary implementation of a proposed product used for validation or exploration.
- **Scenario:** A description of a single instance of a system's usage.
- **Stakeholder:** Any individual or group with a vested interest in the software system being developed.
- **Use Case:** A sequence of interactions between a system and an actor that results in an outcome of value.
- **Validation:** The process of confirming with stakeholders that the documented requirements are correct and complete.

Chapter 5

Requirements Analysis and Modeling: Strategic Briefing

Executive Summary

This document synthesizes the principles and practical applications of Requirements Analysis, focusing on the transition from raw organizational needs to structured Domain and Use-Case Models. The analysis emphasizes that a System Requirements Specification (SRS) must define *what* a system should do without prescribing *how* it should be implemented.

Key takeaways include:

- **Requirements Engineering Process:** A four-stage flow consisting of Elicitation, Analysis, Specification, and Validation.
- **Domain Modeling:** The capture of data requirements by identifying persistent classes, associations, and attributes derived from nouns and verbs in the requirements statement.
- **Use-Case Modeling:** The capture of functional requirements by identifying actors (users or external systems) and grouping system functionalities into meaningful goals.
- **Modeling Fidelity:** High-quality models must exclude implementation constructs (e.g., "Web," "Database," "Login"), avoid over-specification, and focus on persistent rather than transient data.

1. The Requirements Engineering Framework

The Requirements Engineering (RE) process is a structured sequence designed to manage system inception and requirements throughout the development lifecycle.

1.1 The RE Process Flow

The process is categorized into four primary activities:

1. **Elicitation:** Collecting data on system requirements and constraints from domain experts and users.
2. **Analysis:** Understanding the application domain, identifying user needs, defining scope, and determining economic, technical, and operational risks.
3. **Specification:** Documenting requirements in the System Requirements Specification (SRS).
4. **Validation:** Verifying the correctness and completeness of the gathered requirements.

1.2 The System Requirements Specification (SRS)

The SRS serves as the official statement of system requirements. It is a non-design document that focuses exclusively on system behavior.

Method of Writing SRS	Description
Natural Language	Sentences supplemented by diagrams and tables.
Structured Natural Language	Restricted language following a fixed template.
Graphical Notations	UML models combined with structured text annotations.
Mathematical Specs	Notations based on formal mathematical concepts.

2. Domain Modeling: Data Requirements

Domain modeling captures the most important classes and their relationships within an application. It provides a glossary of terms and represents things that exist or events that occur for which data must be stored.

2.1 Identifying Model Elements

- **Classes:** Represent business objects, real-world objects, or events. They typically appear as nouns or noun phrases in requirements (e.g., "Customer," "Movie," "Reservation").
- **Associations:** Represent structural properties and relationships. They typically appear as verbs or verb phrases (e.g., "Customer *purchases* Movie").
- **Attributes:** Correspond to nouns followed by possessive phrases (e.g., "Customer's address"). They represent specific data points like name, age, or price.

2.2 Evaluation Criteria for Domain Models

To ensure a stable system, models should be evaluated against the following standards:

- **Class Validity:** Classes must represent persistent data. Redundant, vague, or implementation-specific classes (e.g., "System," "Database") should be eliminated.
- **Association Integrity:** Associations should describe structural properties, not transient operations. Derived associations—those that can be inferred from other relationships—should be removed to avoid redundancy.
- **Attribute Appropriateness:** Attributes must be closely related to their class. Crucially, **object identifiers (IDs)**should not be included as attributes in a domain model, as they are implementation constructs.

3. Use-Case Modeling: Functional Requirements

Use-case modeling identifies the functional requirements of a system by describing interactions between actors and the system to achieve specific goals.

3.1 Actors and Functionality

- **Actors:** Represent roles played by human users, external hardware, or other systems.

- *Note:* The client organization, internal implementation devices (e.g., “Web,” “Telephone”), and communication methods are **not** actors.

- **Use Cases:** Groups of functionalities that provide something of value to an actor.
  - *Note:* ”Login” is generally considered a non-functional security requirement, not a functional use case.

3.2 Detailed Use-Case Specifications

A detailed specification describes the sequence of events. The sources identify two primary styles for these descriptions:

- **Style 1:** Uses a continuous basic flow with extension points for alternative or exceptional flows.
- **Style 2:** Utilizes **Subflows** to break down complex logic into reusable or more manageable segments.

Core Components of a Specification:

- **Basic Flow:** The “happy path” or standard successful interaction.
- **Alternative/Variant Flows:** Deviations from the basic flow (e.g., “Invalid Term,” “Schedule Exists”).
- **Preconditions and Postconditions:** The state of the system before and after the use case execution.

4. Case Study Synthesis: The Movie Shop System

The Movie Shop System provides a practical application of these modeling principles for managing movie sales, rentals, and reviews.

4.1 Domain Model Analysis

The system distinguishes between a `Movie` (the description/metadata) and a `RentalCopy` (the physical unit).

Class	Key Attributes
Movie	movieId, title, genre, synopsis, releaseYear, sellingPrice, rentalPrice, isPhysical, isDigital
Customer	name, address, phoneNumber, age, sex, email
Member	memberNumber, password (Generalization of Customer)
RentalCopy	copyNumber, returnDate
Review	reviewText, rating, anonymous

Multiplicity and Constraints:

- A `Member` can reserve a maximum of 5 physical movies at one time.
- A `Customer` can rent an unlimited number of physical movies.
- A `RentalCopy` is associated with exactly one `Movie`.

4.2 Use-Case Model Analysis

The system's functionality is grouped into several core use cases, each serving specific actors.

Use Case	Actors	Brief Description
Buy Movie	Member, Sales Clerk	Selecting movies and quantities for purchase.
Rent Movie	Member, Sales Clerk	Managing rentals, recording copies, and processing returns.
Reserve Movie	Member, Sales Clerk	Reserving up to 5 physical movies for future rental.
Manage Reviews	Customer	Browsing or inputting reviews (up to 100 words) and ratings (1-10).
Manage Customer	Member, Sales Clerk	Entering or updating personal/membership information.

Generate Reports	Manager	Producing reports on sales and rental metrics.

5. Common Modeling Pitfalls

The analysis identifies frequent errors that compromise the utility of requirements models.

5.1 Domain Modeling Errors

- **Implementation Bias:** Including "System," "Web," or "Database" as classes.
- **Identity Misuse:** Using "IDs" to relate classes instead of associations.
- **Over-generalization:** Creating unnecessary hierarchies (e.g., separate classes for Physical vs. Digital movies when attributes suffice).
- **Transient Focus:** Modeling operations or actions (e.g., "Browses") as associations rather than structural links.

5.2 Use-Case Modeling Errors

- **Device Actors:** Misrepresenting input/output devices (e.g., "Telephone") as actors.
- **Granularity Issues:** Creating use cases that are too large (obscure goals) or too small (representing individual operations/functions as use cases).
- **Non-functional Confusion:** Representing business rules (e.g., "10% discount") or security constraints (e.g., "Login") as functional use cases.
- **Structure Charting:** Treating the use-case model as a functional decomposition or structure chart rather than a representation of user goals.

Requirements Analysis and System Modeling Study Guide

This study guide provides a comprehensive overview of Requirements Analysis, focusing on Domain Modeling and Use-Case Modeling. It uses the "Movie Shop System" case study to illustrate how raw requirements are synthesized into structured technical models.

1. Fundamentals of Requirements Analysis

Requirements Analysis is a critical phase in the software development lifecycle that bridges the gap between initial elicitation and final validation. It involves understanding the application domain, identifying user needs, and capturing system requirements through specific modeling techniques.

The System Requirements Specification (SRS)

The SRS is the official statement of system requirements. It serves as a definitive guide for what the system must do without prescribing how it should be implemented.

- **Core Characteristics:** It is a requirements document, not a design document.
- **Documentation Methods:**
    - **Natural Language:** Sentences supplemented by tables and diagrams.
    - **Structured Natural Language:** Follows a fixed template or standard.
    - **Graphical Notations:** Uses UML (Unified Modeling Language) combined with structured text.
    - **Mathematical Specifications:** Based on mathematical concepts.

Primary Modeling Activities

1. **Domain Modeling:** Captures data requirements by identifying important classes and their associations.
2. **Use-Case Modeling:** Captures functional requirements by describing interactions between actors and the system.
3. **Nonfunctional Requirements Capture:** Addresses constraints such as performance, security, and specific rules (e.g., a 10% member discount).

2. Domain Modeling

Domain modeling focuses on the "things" that exist or "events" that occur within a system for which data must be stored persistently.

Identifying Model Elements

- **Classes:** Represented by nouns or noun phrases in requirements (e.g., Movie, Customer, Rental Copy).
- **Associations:** Represented by verbs or verb phrases describing relationships (e.g., Customer *purchases* Movie).
- **Attributes:** Usually correspond to nouns followed by possessive phrases (e.g., Movie's *title*, Customer's *address*).

Evaluation Criteria and Constraints

To ensure a stable and coherent system, modelers must evaluate the relevance of every element:

- **Class Validity:** Classes should represent persistent data, not transient operations or implementation constructs (like "Web" or "Database").
- **Multiplicity Constraints:** These define how many instances of one class can be associated with another.
    - *Example:* In the Movie Shop, a Member can reserve at most five physical movies (0..5), but there is no limit on how many movies they can rent (*).
- **Association Classes:** Used when an attribute depends on the existence of an association between two classes.
    - *Example:* The "quantity" of a purchase depends on the association between a Customer and a Movie.

3. Use-Case Modeling

Use-case modeling identifies the functional requirements of a system from the perspective of its users (actors).

Actors in the Movie Shop System

An actor represents a role that a human, hardware device, or another system plays when interacting with the system.

- **Customer:** Browses and enters reviews; receives overdue notices.
- **Member:** A customer who has paid a fee. Can buy/rent movies, reserve movies, and update personal info via the Web.
- **Sales Clerk:** An employee who processes sales, rentals, returns, and updates customer/movie info.
- **Manager:** An employee who can perform clerk duties and generate reports.

Use Case Detailed Specifications

Detailed specifications describe the "flow of activities" between the actor and the system. Two primary styles exist for documentation:

- **Style 1:** Uses a linear basic flow with extension points for alternative flows.
- **Style 2:** Utilizes "Subflows" to break down complex activities (e.g., a separate subflow for "Create Schedule" within a larger use case).

4. Case Study: The Movie Shop System

The Movie Shop system requirements involve managing both physical and digital media, handling memberships, and allowing for customer reviews.

Domain Model Mapping

Requirement	Model Element

Handing physical/digital movies	**Movie Class** (Attributes: isPhysical, isDigital)
Tracking specific rental copies	**RentalCopy Class** (Attributes: copyNumber, returnDate)
Customer membership and logins	**Member Class** (Generalization of Customer; Attributes: memberNumber, password)
Recording purchase quantities	**Purchase Association Class** (Attribute: quantity)
Limiting reservations	**Constraint:** Max-card(Member, Reserves) = 5

Use Case Grouping

Functionality is grouped into high-level use cases to provide value to actors:

- **Buy Movie:** Facilitates selection and payment for movies.
- **Rent Movie:** Manages rental check-outs, returns, and overdue tracking.
- **Manage Reviews:** Allows customers to input ratings and text (up to 100 words) and choose anonymity.
- **Reserve Movie:** Allows members to hold physical copies.
- **Manage Customer:** Handles the entry and updating of personal data (name, address, age, etc.).

Glossary of Key Terms

Term	Definition
Actor	A role that a user, device, or external system plays when interacting with the system.
Association	A relationship between classes that describes a structural property or a persistent link.
Association Class	A class that specifies properties of an association between two other classes.
Attribute	A data value held by the objects in a class (e.g., name, price, or date).
Domain Model	A visual representation of real-world conceptual classes and their relationships.
Generalization	A relationship where a specialized class (subclass) shares the structure and behavior of a more general class (superclass).
Multiplicity	A constraint that specifies how many instances of one class can be linked to a single instance of another class.
Non-functional Requirement	A requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors (e.g., security, discounts).
SRS	System Requirements Specification; the official document stating what the system is required to do.
Subflow	A discrete segment of a use case's flow of events that is broken out for clarity or reuse.
Use Case	A description of a sequence of actions that a system performs to yield an observable result of value to an actor.
Validation	The process of verifying that the system requirements are correct and complete.

Chapter 6

Strategic Briefing: Principles of Excellent Software Requirements Specification

Executive Summary

The quality of software requirements directly impacts project success, timelines, and stakeholder satisfaction. As illustrated by the dialogue between developers and clients, vague or incomplete requirements lead to "best guesses" that necessitate costly rework and cause schedule overruns. This briefing synthesizes the fundamental characteristics of high-quality requirements, the structure of effective Requirements Specification (SRS) collections, and practical guidelines for writing clear, verifiable, and actionable functional requirements.

The core takeaway is that requirements are "good enough" when they allow a team to proceed with design and construction at an acceptable level of risk. Achieving this requires a transition from natural language ambiguity to disciplined, testable, and consistently formatted specifications.

The Requirements Engineering (RE) Process

The development of requirements is a structured flow designed to move from abstract needs to validated technical specifications. The process comprises four primary stages:

1. **Elicitation:** Gathering needs from stakeholders.
2. **Analysis:** Examining and refining gathered information.
3. **Specification:** Documenting the requirements in a formal SRS.
4. **Validation:** Ensuring the documented requirements meet stakeholder needs.

This cycle falls under the broader categories of **Requirements Development** and **Requirements Management**, supported by **Inception Management**.

Characteristics of Excellent Individual Requirements

For a single requirement to be considered high-quality, it must exhibit the following seven attributes:

Characteristic	Description
Complete	Contains all information necessary for the reader to understand it and the developer to implement it. Gaps should be flagged with "TBD" and resolved before implementation.
Correct	Accurately describes a capability that meets a stakeholder need. It must be traceable to a source (user, business rule, etc.) and not conflict with parent requirements.
Feasible	Implementable within the limitations of the system environment and project constraints (time, budget, staff). Prototypes can help evaluate feasibility.
Necessary	Must provide business value, differentiate the product, or satisfy external regulations. If a requirement's inclusion cannot be justified, it should be removed.
Prioritized	Ranked by importance to business value. This allows managers to respond effectively to schedule overruns or resource losses.
Unambiguous	Clear and comprehensible; it must have only one possible interpretation. Peer reviews are essential to catch differing interpretations.
Verifiable	A tester must be able to devise objective tests to determine if it was implemented correctly. If it is a matter of opinion, it is not verifiable.

Characteristics of Requirements Collections (The SRS)

Beyond individual statements, the entire collection of requirements within a Software Requirements Specification (SRS) must maintain its own integrity:

- **Complete:** The SRS must not omit assumed or implied requirements, as these carry higher risk. No "TBDs" should remain in a finalized specification.

- **Consistent:** Requirements must not conflict with one another or with higher-level business and system requirements. Recording the originator of each requirement helps resolve conflicts.
- **Modifiable:** The SRS must be easy to update. This is achieved by maintaining change histories, identifying dependencies between requirements, and using unique labels.
- **Traceable:** Requirements must be linkable backward to their origin and forward to design elements, code, and tests.
- **Design Independence:** The SRS should avoid unnecessarily constraining the developer's design options unless a specific constraint is required (e.g., matching an existing status bar).
- **Granularity:** Requirements should be written at a consistent level, ideally as individually testable units.

Operational Guidelines for Writing Requirements

1. Perspective and Syntax

Requirements can be written from the system's or the user's perspective. Effective communication is the goal, so intermingling these styles is acceptable if it improves clarity.

- **System Perspective:** Use the syntax: `[Optional Precondition] [Optional Trigger Event] the system shall [Expected System Response].`
    - *Example:* "If the chemical is found, the system shall display a list of all containers."
- **User Perspective:** Use the syntax: `The [User Class] shall be able to [Do Something] [To Some Object] [Qualifying Conditions].`
    - *Example:* "The Chemist shall be able to reorder any chemical by editing previous order details."

2. Writing Style and Structure

- **The "Punch Line" First:** State the core need or functionality first, followed by supporting details like rationale or priority.
- **Avoid Creative Writing:** Do not use synonyms for variety (e.g., switching between "client" and "user"). Use consistent terminology defined in a glossary.
- **Active Voice:** Always identify the entity taking the action. Passive voice creates ambiguity regarding who or what performs a task.
- **Use Visuals:** Break up long text with tables, diagrams, and lists to improve communication for different learning styles.

3. Determining the Level of Detail

The required level of detail varies based on the project environment.

- **More Detail Needed:** When work is outsourced, project members are geographically dispersed, or testing is based strictly on the SRS.
- **Less Detail Needed:** When customers are extensively involved, developers have high domain experience, or the project is an internal replacement of an existing system.

Mitigating Ambiguity in Language

Natural language is inherently prone to interpretation errors. The following "fuzzy" constructs must be avoided or clarified:

Ambiguous Terms and Solutions

Ambiguous Term	Recommended Improvement
and/or	Specify "any combination" or define the exact logic to prevent multiple interpretations.

fast, quick, rapid	Specify the maximum acceptable response time (e.g., "within 2 seconds").
user-friendly, easy	Define specific observable characteristics or quantifiable usability metrics.
including, etc.	Provide a complete list or refer to a full list elsewhere; otherwise, the scope is unknown.
reasonably, robust	Explain how a developer or system can judge this condition objectively.
should, probably	Replace with "shall" or "must" to confirm the requirement's necessity.

Structural Ambiguities

- **The A/B Construct:** Slashes (e.g., "Delivery/Fulfillment Team") can mean the team name, "A and B," or "A or B." These should be replaced with explicit language.
- **Boundary Values:** Avoid vague ranges like "5 to 10 days." Use terms like "inclusive," "exclusive," "through", or "longer than" to clarify if endpoints are included.
- **Negative Requirements:** State what the system *shall do* rather than what it *will not do*. For example, "The system shall allow activation only if the contract is in balance" is clearer than "Prevent activation if not in balance."

Case Study: Refinement in Action

Refining flawed requirements often makes them longer because it uncovers missing information.

- **Original:** "The Background Task Manager shall provide status messages at regular intervals not less than every 60 seconds."
- **Issues:** What messages? Where are they displayed? Is 60 seconds the minimum or maximum?
- **Refinement:**
    1. The BTM shall display status messages in a designated UI area. 1.1. The BTM shall update messages every 60 (± 5) seconds. 1.2. The BTM shall display a "Done" message upon completion.

The “Analysis Paralysis” Trap

While requirements should be high-quality, they will never be perfect. The goal is not a flawless document but one that minimizes risk enough to allow the team to move forward. Constant peer review and feedback from those who receive the requirements are the best teachers for achieving this balance.

Requirements Specification and Writing Study Guide

This study guide provides a comprehensive overview of the principles, processes, and best practices for developing high-quality software requirements, based on the provided technical documentation.

Glossary of Key Terms

Term	Definition
Ambiguity	A quality of natural language where a statement can have more than one interpretation, or different readers interpret it differently.
Analysis Paralysis	A state where a team spends excessive time attempting to create perfect requirements, hindering project progress.
BA (Business Analyst)	The individual responsible for resolving conflicts among requirements before they reach developers.
Baseline	A version of requirements that has been officially reviewed and agreed upon, after which change history must be maintained.
Consistent	A requirement that does not conflict with other requirements of the same type or higher-level business and system needs.
EARS Syntax	"Easy Approach to Requirements Syntax"; a structured template for writing functional requirements.
Feasible	The quality of a requirement being implementable within the system's technical limitations and project constraints (time, budget, staff).
Modifiable	A characteristic of an SRS that allows for changes through unique labels, change histories, and cross-references while avoiding redundancy.
Peer Review	A formal review among colleagues to compare interpretations of requirements and catch problems like unvirefability or ambiguity early.
SRS	Software Requirements Specification; the formal document containing the collection of requirements for a project.

TBD	To Be Determined; a flag used to mark gaps in a specification that must be resolved before implementation.
Traceable	The ability to link a requirement backward to its origin and forward to derived requirements, design elements, code, and tests.
Verifiable	A requirement is verifiable if its implementation can be proven through objective testing.

Chapter 7

Software Requirements Specification: Frameworks and Best Practices

Executive Summary

The Software Requirements Specification (SRS) serves as the definitive documented agreement among stakeholders regarding the product to be built. It is the bridge between requirements development—comprising elicitation, analysis, and specification—and the subsequent phases of design, construction, and verification.

Critical takeaways from the requirements process include:

- **The SRS is a Foundational Document:** It defines functional and nonfunctional requirements, system behaviors, and constraints. It is the basis for project planning, testing, and user documentation.
- **Audience Diversity:** A single document must serve a wide range of stakeholders, from developers and project managers to legal staff and subcontractors.
- **Persistence in Labeling:** Unique, persistent identifiers are mandatory for traceability and change management. Hierarchical textual tags or hybrid numbering systems are preferred over simple lists.
- **Managing Incompleteness:** The use of “TBD” (To Be Determined) is a necessary professional standard to flag missing information, provided these items are tracked to closure and resolved before implementation.

- **UI Caution:** While wireframes and prototypes make requirements tangible, they must not replace functional requirements or allow visual design to create functional gaps.

The Requirements Engineering (RE) Process

The RE process is a structured progression designed to transform abstract needs into a concrete specification. It follows a logical flow:

1. **Elicitation:** Gathering needs from stakeholders.
2. **Analysis:** Refining and reconciling those needs.
3. **Specification:** Documenting the agreed-upon requirements.
4. **Validation:** Ensuring the documented requirements accurately reflect stakeholder needs.

This process is supported by **Inception Management**, **Requirements Development**, and **Requirements Management**. The ultimate result is a documented agreement that serves as the "single source of truth" for the product.

The Software Requirements Specification (SRS)

The SRS may be referred to by several names, including the Business Requirements Document (BRD), functional specification, or product specification. Regardless of the name, its core components remain consistent.

Scope of the SRS

- **Functions and Capabilities:** What the system must do.
- **Characteristics and Qualities:** Desired system attributes such as performance, security, and usability.
- **Constraints:** The limits within which the system must operate.

- **System Behaviors:** How the system responds under various conditions.

What the SRS is Not

The SRS should focus on *what* the system should do, rather than *how* it should be built. It should exclude details regarding design, construction, or project management, except where those details act as known implementation constraints.

Stakeholder Audience and Information Needs

The SRS is a multipurpose document. If a desired capability or quality is absent from the requirements agreement, it cannot be expected to appear in the final product.

Stakeholder	Role / Information Need
Developers	Need to know exactly what to build.
Project Managers	Use the SRS for schedule and resource estimations.
Customers/Marketing	Ensure the product aligns with their expectations.
Testers	Develop requirements-based tests and test plans.
Maintenance & Support	Understand the intended function of each product component.
Documentation/Training	Develop user manuals and educational materials.
Legal Staff	Ensure compliance with laws and regulations.
Subcontractors	Base their work on—and are legally held to—specified requirements.

Requirements Labeling Methods

Every requirement requires a unique and persistent identifier to facilitate change requests, modification history, and traceability.

Common Labeling Approaches

Method	Description	Pros	Cons
Sequence Number	Unique numbers (e.g., UC-9, FR-26).	Easy to move within a document while keeping the ID.	No logical grouping; tells nothing about intent.
Hierarchical Numbering	Numbers based on section (e.g., 3.2.4.3).	Familiar and supported by word processors.	Labels can become excessively long; numbers change if sections are moved (non-persistent).
Hierarchical Textual Tags	Structured tags (e.g., Product.Cart.Error).	Meaningful and unaffected by document changes.	Difficult to maintain uniqueness and can be long.

Mitigations for Effective Labeling

- **Hybrid Approach:** Combine hierarchical section numbering with short text codes and sequence numbers (e.g., Section 3.5 Editor Functions: ED-1, ED-2).
- **Textual Sequences:** Use a text tag followed by a sequence number for small sets (e.g., Product.Cart.01).

Dealing with Incompleteness

Missing information should never be ignored. The notation **TBD** (To Be Determined) is used to flag gaps.

- **Resolution Strategy:** Aim to resolve all TBDs before implementation. If work must proceed, defer the specific unresolved portions or design them for easy modification.

- **Management:** TBDs should be numbered, assigned to a responsible party with a deadline, and tracked in an issues list to closure.

User Interfaces and the SRS

Incorporating UI designs in an SRS is a balancing act. While beneficial for clarity, it carries significant risks.

- **Benefits:** Makes requirements tangible through wireframes or mock-ups and sets expectations for "look and feel."
- **Drawbacks:** Increases document size, can delay baselining, and may lead to "design-driven" requirements where functional gaps are overlooked.
- **Core Principle:** Screen layouts do not replace functional requirements. Developers should not be expected to derive data relationships or underlying functionality solely from screenshots.

Readability and Documentation Best Practices

To ensure the SRS is an effective communication tool, the following practices should be implemented:

- **Consistency:** Use a standard template and maintain consistent styles for labels and sections.
- **Visual Emphasis:** Use bold or italics consistently, but avoid relying solely on color (consider grayscale printing and color blindness).
- **Navigation:** Include a Table of Contents, number all figures/tables with captions, and use hyperlinks for cross-references.

- **Requirement Shells:** Use a structured format for individual requirements, such as the **Volere Shell**.

The Volere Shell Components

A comprehensive requirement entry should include:

- **Unique ID and Requirement Type**
- **Description:** A one-sentence statement of intention.
- **Rationale:** Justification for the requirement.
- **Source:** Who raised the requirement.
- **Fit Criterion:** A measurement to test if the solution matches the requirement.
- **Stakeholder Satisfaction/Dissatisfaction:** Scales (1-5) measuring the impact of the requirement's presence or absence.
- **Dependencies and Conflicts:** Other requirements that are linked or incompatible.
- **History:** Record of creation and changes.

Section 4: Glossary of Key Terms

Term	Definition
Business Requirements	High-level objectives and goals usually captured in a vision and scope document.
Elicitation	The first stage of the RE process involving the gathering of requirements from stakeholders.
Fit Criterion	A measurement applied to a requirement to determine if the delivered solution matches the original intent.
Functional Requirements	Descriptions of the behaviors and functions a system must provide under various conditions.
Hierarchical Textual Tags	A labeling method using structured words (e.g., Product.Cart.Error) that are unaffected by moving or deleting requirements.
Nonfunctional Requirements	Descriptions of system qualities such as performance, security, usability, and constraints.
Software Requirements Specification (SRS)	A document stating the functions, capabilities, and constraints of a software system; also known as a BRD or functional specification.
TBD (To Be Determined)	A notation used to flag missing information in a requirements document that must be resolved before implementation.
Traceability Matrix	A tool used to track requirements throughout the project; facilitated by unique and persistent requirement identifiers.
Volere Shell	A standardized template (or “requirement shell”) used to capture detailed information about an individual requirement, including its rationale, source, and satisfaction scale.

Chapter 8

Software Requirements Specification (SRS): Standardized Processes and Structural Frameworks

Executive Summary

Requirements Engineering (RE) is an iterative, multi-stage process essential for defining the scope, boundaries, and essential functions of a software system. The primary artifact of this process, the Software Requirements Specification (SRS), serves as the authoritative basis for contractual agreements between customers and suppliers. Current industry practices have transitioned from the older IEEE 830-1998 standard to the more comprehensive ISO/IEC/IEEE 29148:2011 standard, which provides a unified treatment of requirements throughout the system life cycle.

A high-quality SRS must be feasible, objectively verifiable, and tailored to a diverse audience including customers, analysts, developers, testers, and project managers. Utilizing standardized SRS templates ensures comprehensive coverage of functional requirements, data requirements, external interfaces, and nonfunctional quality attributes. Implementation of these standards reduces development effort by identifying requirements early, thereby minimizing costly redesigns and providing a realistic basis for cost and schedule estimation.

The Iterative Requirements Development Process

Requirements development is not a linear path but an iterative cycle designed to refine understanding and ensure accuracy. The process is comprised of four primary stages connected by feedback loops:

- **Elicitation:** Gathering requirements from stakeholders.
- **Analysis:** Examining elicited information to identify gaps and conflicts. This stage often requires returning to Elicitation to "clarify" findings.
- **Specification:** Documenting the requirements in a formal SRS. Gaps identified here require returning to Analysis to "close gaps."
- **Validation:** Confirming the requirements are correct with stakeholders. This may trigger a "rewrite" of the specification, a "re-evaluate" of the analysis, or a "confirm and correct" loop back to elicitation.

The broader RE process is categorized into **Requirements Development** (Inception and Development) and **Requirements Management**.

Evolution of Requirements Standards

The framework for creating an SRS has evolved to emphasize the entire life cycle of software engineering.

IEEE 830-1998

Historically known as the "IEEE Recommended Practice for Software Requirements Specifications," this standard describes the content and qualities of a good SRS and provides sample outlines.

ISO/IEC/IEEE 29148:2011

This standard superseded IEEE 830. It focuses on the unified treatment of processes and products involved in engineering requirements. Key improvements include:

- Increased emphasis on the characteristics of good requirements.
- Detailed focus on RE activities and operational contexts.
- Definition of five key deliverables:
    1. **StRS:** Stakeholder Requirements Specification
    2. **SyRS:** System Requirements Specification
    3. **SRS:** Software Requirements Specification
    4. **OpsCon:** System Operational Concept
    5. **ConOps:** Concept of Operations

While highly detailed and comprehensive, these deliverables are often time-consuming to produce and are best suited for large-scale software solutions.

The SRS Template: Structural Components

An SRS template acts as a reminder of the knowledge areas that must be explored. While organizations should adopt different templates for different project classes (e.g., new large systems vs. minor enhancements), a standard SRS typically follows a specific eight-part structure.

1. Introduction

The introduction provides a roadmap for the document and defines its purpose.

- **Purpose:** Identifies the product and release number.
- **Document Conventions:** Defines standards or typographical notations used.

- **Project Scope:** Relates the software to business objectives and summarizes major features.
- **References:** Lists resources the SRS refers to, such as contracts or other specifications.

Example Typographical Conventions:

Usage	Typeface	Examples
New terms	Italic	defined by xyz
File and directory names	Italic	C:\derby
SQL examples	Bold and/or fixed-width	SELECT city_name FROM Cities
SQL keywords	All caps	CREATE TABLE

2. Overall Description

This section establishes the context and environment.

- **Product Perspective:** Explains if the product is a new entity, a replacement, or a component of a larger system.
- **User Classes:** Identifies stakeholders and highlights "favored" user classes.
- **Operating Environment:** Details hardware platforms, OS versions, and geographical locations of servers/databases.
- **Constraints:** Lists factors that limit developer options, such as programming languages or code libraries.
- **Assumptions and Dependencies:** Lists external components or business-related assumptions (e.g., assuming a specific library will be reused).

3. System Features

Functional requirements are organized by "features." Each feature block includes:

- **Description and Priority:** A short summary and its current importance level.

- **Functional Requirements:** Itemized capabilities required for the user to perform services.
- **Error Handling:** Responses to invalid input or error conditions.

4. Data Requirements

This section describes how the system processes information.

- **Logical Data Model:** Visual representations like ER diagrams or UML class diagrams.
- **Data Dictionary:** Defines the meaning, data type, length, and format of data elements.
- **Reports:** Describes generated layouts and delivery methods.
- **Data Acquisition and Retention:** Policies for data inventory, backups, and disposal.

5. External Interface Requirements

Ensures the system communicates properly with external elements.

- **User Interfaces:** References to style guides, standards for fonts/icons, screen resolution constraints, and accessibility accommodations.
- **Software Interfaces:** Defines message formats and data mappings between the system and other applications or databases.
- **Hardware Interfaces:** Supported device types and communication protocols.
- **Communications Interfaces:** Requirements for email, web browsers, or network protocols, including security and encryption.

6. Quality Attributes

These are nonfunctional requirements that must be specific, quantitative, and verifiable.

- **Usability:** Ease of learning, efficiency of interaction, and error recovery.

- **Performance:** Specific requirements for system operations.
- **Security:** Privacy issues and access restrictions, often originating from business rules.
- **Safety:** Safeguards against loss, damage, or harm resulting from use.
- **Additional Attributes:** Availability, efficiency, portability, reliability, reusability, and verifiability.

7. Internationalization and Localization

Ensures the product is suitable for different cultures and regions. Requirements address:

- Currency and date formatting.
- Language and character sets.
- Time zones and international laws.
- Physical standards (paper sizes, electrical voltages).

8. Other Requirements

Captures items not covered elsewhere, such as legal, regulatory, or financial compliance, as well as installation and audit trail requirements.

Best Practices for Specification

To ensure the SRS is a functional and effective document, practitioners should adhere to the following guidelines:

- **Avoid Information Redundancy:** Do not duplicate info in multiple sections. Use cross-references and hyperlinks instead.
- **Version Control:** Maintain a revision history recording who made changes, when, and why. This ensures all stakeholders are working from the most current version.

- **Objective Verifiability:** Each requirement must be written so it can be verified via inspection, demonstration, analysis, or testing.
- **Focus on External Behavior:** The SRS should describe what the product does, not how it is designed or the internal production process (which belongs in a separate document).
- **Glossary Management:** Define specialized terms and acronyms. Consider a reusable enterprise-level glossary to maintain consistency across multiple projects.

Part 4: Glossary of Key Terms

Term	Definition
Analysis	The stage in the RE process where requirements are examined to clarify details and close gaps before specification.
ConOps	Concept of Operations; a high-level deliverable proposed by IEEE 29148:2011.
Data Dictionary	A repository containing the meaning, data type, length, format, and allowed values for data elements used by the system.
Design Constraint	Factors that restrict the options available to developers, such as specific programming languages or hardware platforms.
Elicitation	The process of gathering requirements from stakeholders through various methods like notes or interviews.
External Interface	The points of interaction between the software system and users, other software, or hardware components.
Functional Requirement	An itemized list of capabilities the software must implement for the user to carry out services or perform use cases.
Internationalization	Ensuring a product is suitable for use in different nations, cultures, and geographic locations through flexible design.
Localization	The process of adapting a product for a specific local market, including language translation and formatting conventions.
Nonfunctional Requirement	Requirements that specify system attributes such as performance, security, and usability rather than specific behaviors.
OpsCon	Operational Concept; an information item used to describe how a system will be operated in its intended environment.
Quality Attribute	A characteristic of the system (e.g., availability, efficiency, robustness) that is important to customers, developers, or testers.
SRS	Software Requirements Specification; a document that describes the essential requirements and external interfaces of a software product.
StRS	Stakeholder Requirements Specification; a document focused on the needs and requirements of system stakeholders.

SyRS	System Requirements Specification; a document outlining the requirements for the system as a whole.
Validation	The final stage in the RE process where requirements are confirmed and corrected to ensure they meet stakeholder needs.
Verifiable	A characteristic of a requirement stating it must be described such that it can be proven through inspection, demonstration, analysis, or testing.

Chapter 9

Strategic Management of Nonfunctional Requirements in Software Development

Executive Summary

Software success is defined by more than the delivery of requested functionality. While functional requirements describe what a system does, nonfunctional requirements (NFRs)—specifically quality attributes, external interfaces, and constraints—determine how well the system performs and how users perceive its value.

Critical takeaways from this analysis include:

- **The Cost of Omission:** It is significantly more expensive to re-architect a completed system to meet essential quality goals, such as security or performance, than it is to design for them at the outset.
- **Optimization through Trade-offs:** Excellent products result from an optimum balance of competing characteristics. Maximizing one attribute (e.g., security) often necessitates a compromise in another (e.g., performance).
- **Structured Elicitation:** Relying on unstated user expectations for quality is a high-risk strategy. Successful development requires a deliberate, five-step process: using a broad taxonomy, stakeholder selection, prioritization, eliciting specific expectations, and writing SMART requirements.

- **Categorization Fidelity:** Whether a requirement is classified as functional or nonfunctional is less important than ensuring the requirement is identified and documented accurately.

The Nature of Nonfunctional Requirements

Nonfunctional requirements represent the expectations users have regarding a product's operation. These are often unstated but are critical to user satisfaction. A system may meet every functional requirement—such as a heating system that maintains a precise temperature—yet fail if its operation creates unacceptable side effects, such as excessive noise that renders a room unusable for training.

Classes of Nonfunctional Requirements

The documentation identifies three primary classes of NFRs:

1. **Quality Attributes:** Characteristics such as usability, reliability, and performance (often called the "-ilities").
2. **External Interfaces:** Requirements defining how the system interacts with other systems or users.
3. **Constraints:** Restrictions on the design or implementation choices available to developers.

Classification of Quality Attributes

Quality attributes are categorized by whether they are discernible during software execution and which stakeholders they primarily benefit.

External vs. Internal Quality

Category	Definition	Primary Stakeholder	Impact
External Quality	Characteristics discernible through execution.	Users	Direct impact on user satisfaction and delight.
Internal Quality	Characteristics not discernible through execution.	Developers and Maintenance Staff	Indirectly contributes to satisfaction by making the product easier to enhance, correct, and test.

System-Specific Attribute Prioritization

Different project types demand emphasis on different attributes:

- **Embedded Systems:** Performance, efficiency, reliability, robustness, safety, security, and usability.
- **Internet and Corporate Applications:** Availability, integrity, interoperability, performance, scalability, security, and usability.
- **Desktop and Mobile Systems:** Performance, security, and usability.

Key Quality Attributes and Elicitation Strategies

Effective requirements engineering requires asking targeted questions to move beyond vague descriptors like “user-friendly” or “reliable.”

1. Availability

Definition: The measure of planned uptime during which services are fully operational.

- **SMART Example:** “The system shall be at least 95 percent available on weekdays between 6:00 A.M. and midnight Riyadh Time.”

- **Elicitation Focus:** Define what level of performance constitutes being “available” and determine the business consequences of downtime.

2. Integrity

Definition: Preventing information loss and preserving data correctness.

- **SMART Example:** “The system shall protect against the unauthorized addition, deletion, or modification of data.”
- **Elicitation Focus:** Focus on data protection against accidental loss, corruption, or intentional attack. Integrity has no tolerance for error.

3. Performance

Definition: The responsiveness of the system to user inquiries and actions.

- **SMART Example:** “Authorization of an ATM withdrawal request shall take no more than 2.0 seconds.”
- **Elicitation Focus:** Differentiate between “instant” desires and real requirements (e.g., a spell-check vs. a missile guidance system).

4. Reliability

Definition: The probability of the software executing without failure for a specific period.

- **SMART Example:** “No more than 5 experimental runs out of 1,000 can be lost because of software failures.”
- **Elicitation Focus:** Define “critical failure” versus “nuisance” and determine the maximum acceptable duration for the system to remain offline.

5. Robustness

Definition: The degree to which a system functions properly despite invalid inputs, hardware defects, or unexpected conditions.

- **SMART Example:** "If the text editor fails before the user saves, it shall recover contents as of at most one minute prior to the failure."
- **Elicitation Focus:** Identify error conditions the system might encounter and how it should recover gracefully.

6. Safety

Definition: The prevention of injury to people or damage to property.

- **SMART Example:** "The therapeutic radiation machine shall allow irradiation only if the proper filter is in place."
- **Elicitation Focus:** Identify failure modes or operator actions that pose risks to life or limb.

7. Security

Definition: Blocking unauthorized access to functions or data.

- **SMART Example:** "The system shall lock a user's account after four consecutive unsuccessful logon attempts within five minutes."
- **Elicitation Focus:** Explore privilege levels, authentication (biometrics, passwords), and protection against malware.

8. Usability

Definition: Factors constituting "user-friendliness," including ease of learning, memorability, and efficiency of interaction.

- **SMART Example:** "A trained user shall be able to submit a request for a chemical from a vendor catalog in an average of three minutes."
- **Elicitation Focus:** Measure task completion time, error rates, and the number of interactions required to accomplish a task.

9. Internal Quality Attributes (Modifiability, Portability, Scalability)

- **Modifiability:** Addresses how easily code can be understood and changed. *Example:* Revised reports should require 10 hours or less of development effort.
- **Portability:** The effort needed to migrate software to another environment. *Example:* Porting browser bookmarks between Firefox and Chrome.
- **Scalability:** The ability to grow to accommodate more users or data without compromising performance. *Example:* Ability to handle a 30 percent page-view growth per quarter.

Managing Attribute Trade-offs

Attributes often conflict, making it impossible to maximize all of them simultaneously. For example, adding security layers often decreases performance.

Prioritization Method

A spreadsheet-based pairwise ranking comparison is recommended to resolve conflicts:

1. **Elicit:** Identify relevant attributes for the specific system (e.g., for an airport kiosk, usability and security).
2. **Compare:** For each pair, determine which is more important to project success.

3. **Bias Resolution:** Use the resulting scores to bias conflict resolution. If security scores higher (7) than performance (4), the design should favor security when a clash occurs.

Positive and Negative Relationships

- **Negative Impact (-):** Increasing performance or efficiency often adversely affects modifiability, portability, and reusability.
- **Positive Impact (+):** Increasing integrity or reliability often has a positive effect on safety and usability.

Constraints

Constraints are restrictions on design or implementation choices. They differ from quality attributes in that they are often externally imposed and non-negotiable.

Common Sources of Constraints

- **Technology:** Specific tools, languages (e.g., .NET 4.5), or databases that must be used.
- **Environment:** Platform restrictions (e.g., specific web browsers or hardware size/weight).
- **Compatibility:** Backward compatibility with earlier product versions.
- **Compliance:** Regulations, business rules, or legal licenses (e.g., GNU General Public License).
- **Physical:** Hardware limitations (e.g., ATMs only containing $20 bills).

Critical Distinction: Business analysts must distinguish between a user's "requirement" that is actually a solution idea (a constraint) and the underlying functional need. For example, "The

user clicks a button to sort" is a UI constraint that may mask a simpler need for "The system shall provide a way to sort the list."

Specification Standards: The SMART Mnemonic

Quality requirements must be written to avoid subjectivity. Vague terms like "user-friendly" or "24x7 availability" are not measurable or realistic. All NFRs should be:

- **S**pecific
- **M**easurable
- **A**ttainable
- **R**elevant
- **T**ime-sensitive

Section 4: Glossary of Key Terms

Term	Definition
Availability	A measure of planned uptime during which services are fully operational and accessible.
Constraints	Restrictions on the design or implementation choices available to a developer, often imposed by stakeholders or regulations.
Efficiency	An internal quality attribute relating to how well a system uses computer resources; it impacts observed performance.
External Quality	Quality attributes discernible through the execution of software, such as usability and reliability.
Integrity	The prevention of information loss and the preservation of data correctness and protection.
Internal Quality	Quality attributes not discernible during execution, primarily important for maintenance and portability.
Modifiability	The ease with which software designs and code can be understood, changed, and extended.
Nonfunctional Requirements	Expectations about how well a product works, encompassing quality attributes, constraints, and external interfaces.
Performance	The responsiveness of a system to user inquiries and actions under specific conditions.
Portability	A measure of the effort required to migrate software from one operating environment to another.
Quality Attributes	Characteristics such as ease of use, speed, and reliability that distinguish a product beyond its basic functionality.
Reliability	The probability of the software executing without failure for a specific period of time.
Robustness	The degree to which a system functions properly when faced with invalid inputs or unexpected conditions.
Safety	Requirements aimed at preventing a system from causing injury to people or damage to property.
Scalability	The ability of an application to grow to accommodate more users, data, or transactions without compromising performance.
Security	The process of blocking unauthorized access to system functions or data.
Usability	The measure of effort required to prepare input for, operate, and interpret the outputs of a system.

Chapter 10

Strategies and Methodologies for Requirements Prioritization

Executive Summary

Requirements prioritization, frequently referred to as "triage," is a critical phase in project management and software engineering. The necessity of this process arises from a fundamental imbalance: the volume of requirements from diverse stakeholders typically exceeds the limitations of available time, budget, and resources. Without systematic prioritization, projects risk overextending resources on low-value features, leading to missed deadlines and inflated development costs.

Key takeaways from this analysis include:

- **The 80/20 Rule:** 80% of a product's value or revenue is typically generated by only 20% of its functionalities. Identifying this "critical 20%" is the primary goal of prioritization.
- **Trade-off Management:** Prioritization is an exercise in negotiation and compromise, balancing competing goals of business value, implementation cost, and time-to-market.
- **Methodological Rigor:** While small projects may manage priorities informally, large or contentious projects require structured, analytical techniques to mitigate political friction and subjective guesswork.

- **Risk Integration:** Modern risk management (ISO 31000:2009) views risk as the "effect of uncertainty on objectives," encompassing both positive opportunities and negative threats that must be weighted during prioritization.

The Necessity of Requirements Triage

The prioritization process is hindered by several recurring difficulties that necessitate a formal approach.

Primary Challenges

- **Information Overload:** Requirements originate from numerous sources, often resulting in a volume that is impossible to implement simultaneously.
- **Stakeholder Dissonance:** Different stakeholders possess conflicting goals and varying levels of influence. Developers may lack insight into business value, while clients may not grasp technical complexity.
- **Informal Processes:** Organizations often lack systematic metrics, relying instead on ad-hoc or manual prioritization.
- **The "Everything" Fallacy:** A common pitfall is the attitude that all specifications are mandatory. This often leads to "accidental triage," where requirements are discarded haphazardly as deadlines approach.

Objectives of Effective Prioritization

Successful triage must assist in:

1. Making acceptable trade-offs between value, cost, and schedule.
2. Allocating resources based on a requirement's importance to the project as a whole.

3. Determining the specific iteration or release in which a requirement should be included.

Evaluation Criteria and Decision Drivers

To move beyond subjective "gut feelings," projects must evaluate requirements against specific dimensions.

Criterion	Description
Value to Customer/Company	The degree to which the requirement fulfills business objectives or satisfies the user.
Implementation Cost	The financial and resource investment required to develop the feature.
Time-to-Market	The duration required to deliver the requirement.
Technical/Org Complexity	The ease of implementation at both the technical level and the organizational (process) level.
Risk	The effect of uncertainty, including technical risks and missed opportunities.
External Obligations	Mandatory compliance with laws, standards, or patents.

Core Prioritization Techniques

1. Prioritization Scales (Urgency and Importance)

Requirements are categorized based on two primary dimensions:

- **Urgency Scale:**
    - **High:** Mission-critical; required for the next release.
    - **Medium:** Necessary system operations; required eventually, but can wait.
    - **Low:** Functional or quality enhancement; "nice to have."
- **Importance Scale:**

- **Essential:** Product is unacceptable without satisfaction of these requirements.
- **Conditional:** Enhances the product, but the product is acceptable if absent.
- **Optional:** Functions that may or may not be worthwhile.

2. Wiegers' Semi-Quantitative Approach

This technique uses a formula to derive a numerical priority based on relative weights. It requires stakeholders to estimate four dimensions on a scale (typically 0–9):

1. **Relative Benefit:** Value gained if the requirement is included.
2. **Relative Penalty:** Loss suffered if the requirement is excluded.
3. **Relative Cost:** The implementation effort.
4. **Relative Risk:** Technical or other uncertainties.

The Priority Formula: `Priority = (Value %) / ((Cost % * Cost Weight) + (Risk % * Risk Weight))`

This method removes emotion by forcing a mathematical comparison of relative value against relative cost and risk.

3. The Kano Model (Customer Perception)

The Kano Model groups requirements based on how they impact customer satisfaction versus the level of execution.

- **Basic Requirements:** Taken for granted by the customer (must-be).
- **Performance Requirements:** Explicitly requested; satisfaction increases as execution improves.

- **Excitement Requirements:** Not requested or expected; these provide high satisfaction if included.
- **Indifferent:** The customer does not care if these are present or absent.
- **Reverse:** Features that actually cause dissatisfaction if included.

The Kano Survey Method: Customers are asked two questions for each requirement:

1. **Functional:** "How do you feel if this requirement is included?"
2. **Dysfunctional:** "How do you feel if this requirement is NOT included?"

4. Three-Level Scale (Matrix)

This iterative approach uses a four-quadrant matrix to map **Importance** against **Urgency**.

- **High Priority:** Both Important and Urgent. These must be in the next release or it is considered "unshippable."
- **Medium Priority:** Important but Not Urgent.
- **Low Priority:** Not Important and Not Urgent.
- **The "Waste" Quadrant:** Urgent but Not Important. These should be scrubbed or set to low priority, as they consume resources without adding significant business value.

For large projects, this process is performed **iteratively**. If the "High Priority" group is too large, it is run through the matrix again to subdivide it into Highest, Higher, and High classes.

Specialized and Commercial Tools

While many organizations use editable Excel documents (such as the Volere Prioritisation Analysis), complex projects may utilize dedicated decision-support tools.

- **TeamBLUE Focal Point:** A commercial tool that supports portfolio management and pairwise comparisons of features. It uses dynamic algorithms to reduce the number of comparisons needed and detects inconsistencies in stakeholder responses.
- **Additional Techniques:**
    - **MoSCoW:** (Must have, Should have, Could have, Won't have).
    - **$100 Method:** Stakeholders "spend" a hypothetical $100 budget across requirements.
    - **Analytic Hierarchy Process (AHP):** A mathematical approach for multi-criteria decision-making.

Conclusion

Requirements prioritization is not a one-time event but an iterative process that must be integrated into the Software Requirements Specification (SRS). For industry adoption, the chosen process must be fast, simple, and capable of yielding reliable results. Ultimately, the most effective prioritization strategy identifies the 20% of functionalities that provide 80% of the value, ensuring that limited resources are focused on the most impactful goals.

Comprehensive Study Guide: Requirements Prioritization and Triage

This study guide provides a detailed overview of requirements prioritization, commonly referred to as "triage." It examines the methodologies, challenges, and analytical techniques used to determine which requirements should be included in a product release based on value, cost, and risk.

1. Overview of Requirements Prioritization

Requirements prioritization is the process of deciding which requirements are most important and should be implemented in the current or future releases. It is essential because resources—such as budget and time—are limited, while the number of requirements is often excessive.

The 80-20 Rule

In the context of software development, the 80-20 rule suggests that 80% of the value or results come from 20% of the causes or effort.

- **Application:** 20% of functionalities typically provide 80% of the revenue.
- **Risk of Neglect:** The remaining 80% of functionalities often offer a lower return on investment (ROI) while increasing development costs, maintenance costs, and delivery delays.
- **Goal:** Prioritization aims to identify that most beneficial 20% of functionalities.

Core Challenges

- **Source Volume:** Requirements arrive from many different sources and stakeholders.
- **Stakeholder Conflict:** Different stakeholders have conflicting goals; some decisions carry more weight than others.
- **Knowledge Gaps:** Developers may not understand business value, while clients may not grasp implementation complexity.
- **Lack of Data:** Companies often lack systematic metrics and rely on informal, manual, or ad-hoc processes.

- **The "Everything" Fallacy:** Some stakeholders may insist on implementing every requirement in the specification without acknowledging the associated cost or time constraints.

2. Prioritization Techniques

Technique 1: Prioritization Scales

This method involves categorizing requirements using defined dimensions such as Urgency and Importance.

Dimension	Category	Description
Urgency	High	Mission-critical; required for the next release.
	Medium	Supports necessary operations; could wait for a later release.
	Low	Functional/quality enhancement; "nice to have" if resources permit.
Importance	Essential	Product is unacceptable if these are not satisfied.
	Conditional	Enhances the product, but it is acceptable if absent.
	Optional	Functions that may or may not be worthwhile.

Technique 2: Wiegers' Prioritization

A semi-quantitative approach that calculates priority based on relative benefit, penalty, cost, and risk.

- **Relative Benefit:** The value gained by having the requirement.
- **Relative Penalty:** The consequence to the stakeholder if the requirement is excluded.
- **Relative Cost:** The resources required to implement the requirement.
- **Relative Risk:** The technical or other risks associated with the requirement.

Priority Formula: `Priority = (Value %) / ((Cost % * Cost Weight) + (Risk % * Risk Weight))`

Technique 3: Kano Surveys

The Kano Model groups requirements based on customer perception to maximize satisfaction.

- **Basic:** Taken for granted; if missing, customers are very dissatisfied.
- **Performance:** Specifically requested; satisfaction is proportional to the level of execution.
- **Excitement:** Not requested or expected; provides very high satisfaction when present.
- **Indifferent:** Customers do not care about these requirements.
- **Reverse:** Features that customers actually dislike.

Kano Questioning Method: Customers are asked two questions for each requirement:

1. **Functional:** "How would you feel if this requirement were included?"
2. **Dysfunctional:** "How would you feel if this requirement were NOT included?"

Technique 4: Three-Level Scale

Requirements are mapped on a 2x2 matrix based on Importance and Urgency.

- **High Priority:** Both Important and Urgent. These must be in the next release or the product is not shippable.
- **Medium Priority:** Important but Not Urgent.
- **Low Priority:** Not Important and Not Urgent.
- **The Fourth Quadrant (Urgent but Not Important):** These should be ignored or scrubbed, as they consume time without adding sufficient business value.

5. Glossary of Key Terms

- **80-20 Rule (Pareto Principle):** The concept that 80% of effects (e.g., revenue, results) come from 20% of causes (e.g., effort, functionalities).
- **Basic Requirements (Kano):** Features that customers take for granted; their absence causes extreme dissatisfaction, but their presence does not significantly increase satisfaction.
- **Excitement Requirements (Kano):** Features that are unexpected and unrequested by the customer but provide high satisfaction if included.
- **Importance:** A prioritization dimension measuring how essential a capability is to the customer or the product's acceptability.
- **ISO 31000:2009:** An international standard for risk management that defines risk as the effect of uncertainty on objectives.
- **Pairwise Comparison:** A prioritization method (often used in tools like Focal Point) where requirements are compared in pairs to determine their relative importance.
- **Requirements Triage:** The process of prioritizing requirements to determine which ones will be included in a specific release, typically involving negotiation and compromise.
- **Relative Risk:** In Wiegers' model, an estimation of the technical or other difficulties associated with implementing a requirement.
- **Urgency:** A prioritization dimension based on how quickly a customer needs a specific capability, often tied to the next release cycle.
- **Wiegers' Prioritization:** A semi-quantitative prioritization technique that uses a specific formula to calculate priority based on benefit, penalty, cost, and risk.

Chapter 11

Briefing Document: Specifying Data Requirements in Software Engineering

Executive Summary

Data requirements are a fundamental component of software development that pervade all levels of the three-level requirements model. While functional requirements define what a system does, data requirements define the information the system creates, modifies, processes, and deletes. Failure to define these requirements with discipline leads to significant technical risks, including data overwriting, unterminated strings, and system crashes.

The primary tools for managing these requirements are the **Data Model**, which provides a high-level view of relationships, and the **Data Dictionary**, a shared repository providing granular definitions of data elements. Utilizing these tools ensures a common understanding among team members, reduces integration errors, and facilitates the reuse of data definitions across applications.

1. The Critical Importance of Data Requirements

Data requirements are present anywhere there are system functions. Software functionality is specifically designed to create, modify, display, delete, process, and use data. Examples of data include pixels in video games, packets in cell phone calls, and bank account activity.

Risks of Poor Data Specification

Without a disciplined approach to data management, development teams face several “bad things”:

- **Variable Inconsistency:** Different developers may inadvertently use different names, lengths, or validation criteria for the same data item.
- **Data Corruption:** Interconverting data of different lengths can lead to overwriting other data or picking up stray pad characters.
- **System Failure:** Improperly handled data lengths can result in unterminated character strings or the overwriting of program code, eventually causing the application to crash.

2. The Requirements Engineering (RE) Process

Requirements development is an iterative process. A Business Analyst (BA) or Requirements Engineer (RE) should begin collecting data definitions as they emerge during the elicitation phase.

Iterative Development Cycle

The process flows through four main stages, with continuous feedback loops:

1. **Elicitation:** Gathering raw data needs.
2. **Analysis:** Refining high-level data elements from system context diagrams (input/output flows).
3. **Specification:** Documenting precise definitions.
4. **Validation:** Confirming and correcting data requirements with stakeholders.

3. Modeling Data Relationships

A data model depicts the system's data relationships at a high level of abstraction, while the data dictionary provides the necessary detail.

Entity-Relationship Diagrams (ERD)

The ERD represents logical groups of information (entities) and their interconnections. It helps communicate data components without necessarily implying a specific database implementation.

- **Entities:** Represent physical items (people, objects) or aggregations of data. They are shown as rectangles and named with singular nouns.
- **Attributes:** Describe entities (e.g., a "Chemical" entity has attributes like identifier, name, and structure).
- **Relationships:** Represented by diamonds, these identify logical linkages between entities (e.g., a "Requester" *places* a "Chemical Request").

Notations and Cardinality

Cardinality (or multiplicity) defines the numerical nature of relationships:

- **One-to-One (1:1):** e.g., one Chemical Container tracked by one Container History.
- **One-to-Many (1:M):** e.g., one Requester placing many Chemical Requests.
- **Many-to-Many (M:M):** e.g., many Chemicals listed in many Vendor Catalogs.

Notation Style	Key Visual Elements
Peter Chen	Rectangles (Entities), Diamonds (Relationships), Numbers/Letters (Cardinality)
James Martin	Crow's foot symbol for "many," vertical line for "1," and circle for "0."
UML Class Diagrams	Classes (Entities), Attributes, and Operations (Behaviors).

4. The Data Dictionary

The data dictionary is a collection of detailed information about every data entity used in an application. It serves as the bridge between requirements and implementation (database schemas and variable names).

Benefits of a Shared Repository

- **Validation:** Identifies the criteria for data validation.
- **Integration:** Minimizes errors when different system components interface.
- **Quality:** Prevents redundancies and ensures all team members have a consistent understanding of the data.
- **Efficiency:** Entries are organized alphabetically for easy navigation.

Types of Data Elements

1. **Primitive:** A data element that cannot be further decomposed (e.g., Request ID).
2. **Structure:** Composed of multiple data elements (e.g., "Requester" is a structure containing Name, Employee Number, and Department). Elements are typically separated by a `+` sign.
3. **Repeating Group:** When multiple instances of an element appear in a structure. These are enclosed in curly braces `{min:max}`. An "n" is used if the maximum is unlimited (e.g., `1:n`).

5. Implementation and Validation Considerations

Defining data requirements requires extreme precision to ensure developers can write correct code.

Data Element Specification Table (Example)

Data Element	Composition / Data Type	Length	Values / Validation
Request ID	Integer	8	System generated, sequential
Quantity Units	Alphabetic characters	10	grams, kilograms, etc.
Requester Name	Alphabetic characters	40	Blanks, hyphens, periods allowed

Critical Design Questions

Effective specification requires answering detailed questions during analysis:

- **Case Sensitivity:** Is “Karl” the same as “karl”?
- **Character Sets:** Are diacritical marks (e.g., tilde, umlaut, accent) or non-English alphabets permitted?
- **Formatting:** How should timestamps and dates be displayed?

By maintaining an up-to-date and comprehensive data dictionary, organizations can turn data definitions into reusable assets across various applications within a product line.

Study Guide: Specifying Data Requirements and Modeling

This study guide provides a comprehensive overview of the principles and practices involved in specifying data requirements within software engineering. It focuses on the use of data dictionaries, entity-relationship diagrams (ERDs), and the iterative nature of the requirements engineering process to ensure data consistency and system quality.

Short-Answer Quiz

1. What is a data dictionary, and what primary benefit does it provide to a development team? A data dictionary is a shared repository that defines the meaning, composition, data type, length, format, and allowed values for every data element used in an application. Its primary benefit is providing a disciplined way to manage data, which eliminates problems such as inconsistent variable names, conflicting validation criteria, and system crashes caused by interconverting data of different lengths.

2. How do data requirements relate to the three-level requirements model? While data requirements are not always explicitly shown as a separate level, they pervade all three levels of

the requirements model. Anywhere a function exists, data is present—such as pixels in a game or sales figures in a report—meaning software functionality is specifically designed to create, modify, display, delete, or process that data.

3. Distinguish between a data model and a data dictionary. A data model, such as an entity-relationship diagram, provides a high-level view of the system's data and the relationships between various entities. In contrast, a data dictionary provides the detailed view, defining the specific attributes and precise definitions for the data components identified in the model.

4. According to the Requirements Engineering (RE) process, how do elicitation and specification interact? The RE process is iterative, meaning that information flows back and forth between stages to ensure accuracy. Elicitation results are analyzed, and if gaps are found during specification, the team must return to the analysis or elicitation phases to close those gaps or rewrite requirements for better clarity.

5. What is the significance of "cardinality" in an entity-relationship diagram? Cardinality, also known as multiplicity, defines the numerical relationship between two entities, showing how many instances of one entity can be associated with another. Common cardinality types include one-to-one, one-to-many (such as one requester placing many chemical requests), and many-to-many.

6. How does the James Martin notation differ from Peter Chen notation in representing cardinality? While both use rectangles for entities, Peter Chen notation uses diamonds to show relationships and letters/numbers for cardinality. James Martin notation places relationship labels directly on the connecting lines and uses specific symbols like a vertical line for "one," a crow's foot for "many," and a circle to indicate "zero."

7. In the context of a data dictionary, what defines a "primitive" data element? A primitive data element is a basic unit of data for which no further decomposition is possible or necessary. These elements are described in the dictionary using attributes like data type, length, numerical range, and a list of allowed values.

8. What is a "repeating group" in data composition, and how is it notationally represented? A repeating group occurs when multiple instances of a specific data element can appear within a data structure. It is represented by enclosing the element in curly braces and indicating the minimum and maximum number of possible repeats, such as "{min:max}".

9. Why is the precise definition of data elements like "Requester Name" essential for developers? Precise definitions are essential because they dictate the validation criteria the developer must implement, such as case sensitivity and allowed characters (e.g., hyphens or apostrophes). Without these details, the developer cannot know whether to reject certain inputs or how to handle diacritical marks and specific text formats.

10. How do UML class diagrams serve teams using object-oriented development methods for data modeling? UML class diagrams show data attributes for individual classes and the logical links and cardinalities between them, similar to an ERD. In data modeling specifically, the bottom section of the class rectangle—which usually holds operations or behaviors—is typically left empty.

Glossary of Key Terms

Term	Definition
Attribute	A characteristic that describes an entity; individual instances of an entity have different attribute values.
Cardinality	Also known as multiplicity; the number of instances of one entity that can be logically linked to another (e.g., 1:1, 1:M, M:M).
Crow's Foot	A symbol used in James Martin notation to indicate a cardinality of "many."
Data Dictionary	A shared repository that defines the meaning, composition, type, length, format, and allowed values for all data elements.
Data Model	A high-level depiction of a system's data relationships, often represented via an ERD.
Data Structure	A data element composed of multiple other data elements or structures, usually separated by a "+" in definitions.
Entity	A physical item, person, or aggregation of data represented as a singular noun in a rectangle within an ERD.
ERD	Entity-Relationship Diagram; a tool used to represent logical groups of information and their interconnections.
James Martin Notation	An ERD modeling style that uses symbols on lines (like circles and crow's feet) to represent cardinality.
Peter Chen Notation	A specific notation for ERDs that uses rectangles for entities and diamonds for relationships.
Primitive Data Element	A data element that cannot be further decomposed.
Repeating Group	A data element that can appear multiple times within a structure, denoted by curly braces `{ }`.
UML Class Diagram	A diagram used in object-oriented development to show classes, their attributes, and the cardinalities of the links between them.

Chapter 12
Requirements Validation: Process, Techniques, and Strategic Implementation

Executive Summary

Requirements validation is the final stage of requirements development, serving as a critical quality gate before project baselining. It focuses on ensuring that requirements are correct, demonstrate high quality, and satisfy customer needs. The core distinction lies between **validation** (building the right product) and **verification** (building the product right).

The most impactful takeaway from the analysis is the high Return on Investment (ROI) associated with formal validation techniques. Organizations investing in formal requirements inspections have reported saving up to 10 hours of labor for every hour invested—a 1,000% return. By employing a combination of peer reviews, prototyping, and early testing, development teams can identify ambiguities and omissions before they escalate into costly downstream defects.

1. Defining Validation and Verification

Requirements Engineering (RE) distinguishes between validation and verification, though both are complementary quality assurance processes that must be performed at every stage of development.

1.1 Validation

- **Primary Question:** "Are we building the right product?"
- **Focus:** Assesses whether requirements trace back to business objectives and ensure stakeholder satisfaction.

- **Activity:** Checking with elicitation sources and stakeholders to confirm the system will meet their actual needs.

1.2 Verification

- **Primary Question:** "Are we building the product right?"
- **Focus:** Determines if requirements possess the desirable properties of high-quality specifications (e.g., clarity, lack of ambiguity, and conformity to standards).
- **Activity:** Checking the SRS against writing rules and ensuring consistency between different models.

2. Peer Review Techniques

Peer reviews occur whenever someone other than the author examines a work product for problems. These range from informal feedback sessions to highly structured inspections.

2.1 Informal Peer Reviews

Informal reviews are useful for education and unstructured feedback but lack systematic rigor. Common approaches include:

- **Peer Deskcheck:** One colleague reviews the work.
- **Passaround:** Multiple colleagues examine a deliverable concurrently.
- **Walkthrough:** The author describes the deliverable and solicits comments.

Note: Informal reviews often fail to catch ambiguities because individual reviewers may interpret a requirement differently without realizing others have a conflicting understanding.

2.2 Formal Peer Reviews: Inspections

Developed by Michael Fagan at IBM, inspections are the "best-established" formal review type. They serve as a quality gate that project deliverables must pass before baselining.

Inspection Participants and Roles:

Role	Responsibility
Author	Created the work product; addresses identified defects.
Moderator	Plans the inspection, leads the meeting, and ensures follow-up.
Reader	Leads the team through the document, describing requirements in their own words.
Recorder	Captures identified defects and issues in an action item list.

Key Perspectives for the Inspection Team:

- **Sources:** People who provided the original information.
- **Implementers:** Developers who will build the product based on the requirements.
- **Testers:** Individuals who can spot unverifiable requirements.
- **Interface Representatives:** Those responsible for affected external systems.

3. The Inspection Lifecycle

A successful inspection follows a well-defined multistage process governed by entry and exit criteria.

3.1 Stages of Inspection

1. **Planning:** Author and moderator determine participants and materials.
2. **Preparation:** Inspectors examine the product individually using defect checklists to identify issues. Up to 75% of defects are found during this stage.
3. **Inspection Meeting:** The team identifies defects; the meeting should not exceed two hours.
4. **Rework:** The author resolves the identified defects.
5. **Follow-Up:** The moderator ensures all issues are resolved and exit criteria are met.

3.2 Criteria for Success

- **Entry Criteria:** The document must conform to templates, have line numbers, have "TBD" (To Be Determined) sections clearly marked, and pass a ten-minute "sanity check" by the moderator (no more than three major defects in a sample).
- **Exit Criteria:** All issues must be addressed, changes must be verified as correct, and any remaining open issues must have a documented owner and target date.

4. Prototyping and Testing as Validation

Beyond reviews, requirements must be "realized" through prototyping and testing to ensure they are feasible and complete.

4.1 Prototyping

Prototypes make requirements tangible for stakeholders:

- **Paper Mock-ups:** Used to walk through use cases or functions to detect omissions.
- **Proof-of-Concept:** Demonstrates technical feasibility.
- **Evolutionary Prototypes:** Allow users to see how requirements function in a live environment.

4.2 Early Testing

Designing tests is a powerful validation tool that should begin as soon as user requirements are defined.

- **Conceptual Testing:** Deriving tests from user requirements early in the process reveals ambiguities.
- **The Synergistic Relationship:** If a tester cannot describe the expected system response, the requirement is likely vague.

- **Tracing:** Mapping tests to functional requirements ensures no requirement is overlooked.

Example: Chemical Tracking System (CTS) Tracing a test through a "dialogue map" can identify missing logic. For instance, if a navigation line like "order new container" is never exercised by a test, it either means the navigation is unauthorized or a necessary test is missing.

5. Challenges and Mitigation Strategies

Validation can be hindered by scale, geography, and human factors.

Challenge	Mitigation Strategy
Large Documents	Perform incremental reviews; use risk analysis to prioritize critical or novel sections for formal inspection.
Large Teams	Limit inspection teams to seven or fewer. Use parallel small teams for large requirements sets and combine their defect lists.
Geographically Separated Teams	Use web conferencing and collaborative tools to track body language and ensure all reviewers are viewing the same content.
Unprepared Reviewers	Use strict entry criteria; the moderator should cancel meetings if preparation has not been performed.

6. Requirements Validation Best Practices

- **Standardize with Checklists:** Develop defect checklists for different document types to help reviewers internalize frequent problems (e.g., completeness, correctness, and traceability).
- **Identify Acceptance Criteria:** Define early how users will determine if the solution meets their needs (e.g., passing specific tests, meeting nonfunctional requirements).
- **Simulate:** Use commercial tools to simulate system behavior where possible.
- **Schedule Re-reviews Sparingly:** No individual should review the same material more than three times to avoid fatigue and oversight.

Part 4: Glossary of Key Terms

Term	Definition
Acceptance Criteria	The conditions used by stakeholders to determine if a solution meets their needs, often including passing specific tests and meeting nonfunctional requirements.
Author	The person who created the requirements work product being examined during a review or inspection.
Baseline	A version of the requirements that has passed a quality gate (like a formal inspection) and is formally approved as a basis for further development.
Defect Checklist	A tool used during reviews to call attention to historically frequent requirement problems, such as ambiguities, gaps, or inconsistencies.
Dialogue Map	A visual representation of the navigation and interactions within a system, used to trace test paths and identify missing system behaviors.
Entry Criteria	A set of conditions (e.g., lack of typos, unique identifiers) that a document must meet before a formal inspection can begin.
Exit Criteria	Requirements that must be satisfied to conclude an inspection, such as resolving all raised issues and correcting errors in related work products.
Formal Inspection	A well-defined, multi-stage peer review process (originally developed by Michael Fagan) involving a small team of participants looking for defects.
Moderator	The individual responsible for planning the inspection, leading the meeting, and ensuring follow-up actions are completed.
Peer Deskcheck	An informal review where the author asks a single colleague to look over their work product for errors.

Prototype	A partial or mock-up implementation of a system (paper, digital, or evolutionary) used to validate requirements with users.
Reader	An inspection role responsible for paraphrasing the requirements to the team during the meeting to ensure a common understanding.
Recorder	An inspection role responsible for capturing identified defects and action items in a list for the author to address.
Requirement Validation	The process of ensuring that requirements accurately describe the system capabilities that will satisfy stakeholder needs.
Requirement Verification	The process of determining if requirements are written correctly and possess the characteristics of high-quality requirements.
SRS (Software Requirements Specification)	A document that captures the complete set of functional and nonfunctional requirements for a system.
Walkthrough	An informal review where the author describes a work product to a group and solicits their comments and feedback.

Source Fidelity Note: *This guide is based exclusively on the provided excerpts from the "Requirements Validation" presentations.*

Chapter 13

Requirements Management: A Comprehensive Briefing

Executive Summary

Requirements management (RM) encompasses all activities necessary to maintain the integrity and accuracy of requirements agreements throughout a project's lifecycle. Its primary objective is to ensure that the significant effort invested in requirements development is not lost and to minimize the "expectation gap" between stakeholders and the development team. Effective RM provides a clear roadmap for project progress, identifying the current state of requirements and defining the criteria for completion.

The core pillars of requirements management include:

- **Version Control:** Systematically identifying and tracking changes to requirements and document sets.
- **Change Control:** Implementing a formal process for proposing, evaluating, and deciding on changes to established baselines.
- **Status Tracking:** Monitoring the lifecycle of individual requirements from proposal to verification.
- **Requirements Tracing:** Defining links between requirements and other system elements to ensure all needs are met and no unnecessary work is performed.

1. The Foundations of Requirements Management

Requirements management is distinct from requirements development; while the latter is an iterative process of elicitation, analysis, specification, and validation, the former focuses on maintaining those requirements once they are defined.

1.1 The Necessity of RM

Without disciplined management, projects face several critical risks:

- **Wasted Effort:** Team members may implement features that were previously canceled or modified if they are working from outdated documents.
- **Inaccurate Representation:** If developers implement changes directly in code without updating documentation, the requirements no longer accurately represent the product.
- **Expectation Gaps:** Stakeholders may be misinformed about the current state of the product, leading to dissatisfaction upon delivery.

1.2 Requirements Baselines

A requirements baseline is a set of requirements agreed upon by stakeholders, typically defining the contents of a specific release or iteration.

- **Composition:** It may include business, user, functional, and nonfunctional requirements, as well as data dictionaries and analysis models.
- **Control:** Once a set of requirements is baselined—usually following review and approval—it is placed under formal change management. Subsequent modifications can only occur through defined change control procedures.

2. Version Control and Attributes

Version control must begin as soon as a requirement or document is drafted to maintain a complete history of changes.

2.1 Implementing Version Control

Effective version control requires that every team member has access to the most current version of the requirements. Best practices include:

- **Standardized Identification:** Uniquely identifying versions of individual requirements and requirements sets.
- **Designated Updates:** Permitting only specific individuals to update requirements to minimize confusion.
- **Revision History:** Every version or requirement should track what was changed, the date of the change, the individual responsible, and the rationale.
- **Methodology:** Approaches range from manual labeling and word processor revision marks to the use of dedicated Requirements Management tools, which provide the most robust history tracking and reversion capabilities.

2.2 Requirement Attributes

Beyond the text of the requirement itself, RM tools allow for the tracking of various attributes that provide context and metadata:

- **Creation Data:** Date created, author, and source/origin.
- **Management Data:** Current version number, priority, release number, and status.
- **Technical Context:** Rationale behind the requirement, contact stakeholders, and validation methods/acceptance criteria.

3. Requirements Status Tracking

Tracking the status of functional requirements provides a precise gauge of project progress, moving beyond vague “percentage complete” estimates to objective data.

3.1 Suggested Requirement Statuses

Status	Definition
Proposed	Requested by an authorized source but not yet accepted.
In Progress	A business analyst is actively crafting the requirement.
Drafted	The initial version has been written.
Approved	Analyzed, impact estimated, and allocated to a baseline.
Implemented	Code is designed, written, and unit tested.
Verified	Satisfied acceptance criteria and confirmed via testing/review.
Deferred	Approved but planned for a later release.
Deleted	Removed from the baseline; requires explanation.
Rejected	Proposed but never approved for implementation.

4. Change Control Process

Change is inevitable due to evolving market opportunities, shifting regulations, and business needs. A sensible change control process allows leaders to make informed business decisions regarding cost, schedule, and value.

4.1 Change Control Policy

An organization should communicate clear expectations, including:

- **Universal Adherence:** All changes must follow the process; unsubmitted requests will not be considered.
- **Impact Analysis:** Required for every change to understand consequences before approval.
- **CCB Authority:** A Change Control Board (CCB) decides which changes to implement.
- **Transparency:** The change database must be visible to all stakeholders.

- **No Pre-emptive Work:** No design or implementation work is performed on unapproved changes (excepting feasibility studies).

4.2 Change Request Life Cycle

A change request typically moves through several states:

1. **Submitted:** Originator submits the request.
2. **Evaluated:** Impact analysis is performed regarding feasibility, cost, and alignment.
3. **Approved/Rejected:** The CCB makes a decision based on the evaluation.
4. **Change Made:** A modifier implements the change in the affected work products.
5. **Verified:** Peer reviews or tests confirm the change correctly addresses the request.
6. **Closed:** The process is complete, and modified products are stored correctly.

5. Requirements Tracing

Tracing allows stakeholders to follow the life of a requirement forward to implementation and backward to its origin.

5.1 Trace Link Types

- **Forward to Requirements:** From customer needs (business objectives, market demands) to specific requirements.
- **Backward from Requirements:** From requirements back to customer needs to ensure alignment and necessity.
- **Forward from Requirements:** From requirements to downstream work products (design, code, tests).
- **Backward to Requirements:** From work products back to requirements to ensure all code and tests are justified.

5.2 The Requirements Traceability Matrix (RTM)

The RTM is a table that captures the relationships between different system elements. It can define various cardinalities:

- **One-to-one:** E.g., one design element implemented in one code module.
- **One-to-many:** E.g., one functional requirement verified by multiple tests.
- **Many-to-many:** E.g., shared design elements satisfying several requirements, or use cases leading to multiple functional requirements.

5.3 Benefits of Tracing

- **Identifying Gaps:** Finding business requirements with no corresponding user requirements (missing) or functional requirements with no source (unnecessary).
- **Impact Analysis:** Ensuring no system element is overlooked when a requirement is modified.
- **Maintenance:** Facilitating updates when corporate policies or regulations change.
- **Testing:** Pointing developers toward likely defect areas when tests fail.

6. Requirements Management Best Practices

To ensure project success and requirements stability, the following practices are recommended:

- Establish a formal change control process and a Change Control Board.
- Perform rigorous impact analysis on all proposed changes.
- Establish baselines and maintain strict version control of requirements sets.
- Maintain a detailed history of changes and track the status of every requirement.
- Utilize a Requirements Traceability Matrix to manage interconnections.

- Measure change activity (e.g., total requests, volatility trends) to assess stability; a sustained high frequency of changes may indicate incomplete original requirements or poor elicitation practices.
- Employ a dedicated Requirements Management tool to automate and facilitate these activities.

Comprehensive Study Guide: Requirements Management

Requirements management is a critical discipline within software engineering that encompasses all activities required to maintain the integrity and accuracy of requirements agreements throughout a project's lifecycle. It acts as a bridge between requirements development and the final delivery, ensuring that the investment made in eliciting and analyzing requirements is not lost during implementation.

1. Foundations of Requirements Management

The Requirements Baseline

A requirements baseline represents a set of requirements that stakeholders have formally agreed upon. This baseline often defines the specific contents of a planned release or development iteration.

- **Contents:** It can include business, user, functional, and nonfunctional requirements, as well as data dictionaries and analysis models.
- **Control:** Once a set of requirements is baselined, it is placed under change management. Subsequent modifications can only occur through defined change control procedures.

- **Strategic Options:** To accommodate changes to a baseline, projects may defer lower-priority items, obtain more staff, extend schedules, or, in some cases, sacrifice quality.

Version Control

Version control involves uniquely identifying different versions of individual requirements and requirement sets (typically documents).

- **Best Practices:** Version control should begin as soon as a requirement is drafted. It requires that only designated individuals be permitted to update requirements and that the version identifier changes with every update.
- **Documentation:** Each version should include a revision history identifying the change, the date, the individual responsible, and the rationale.
- **Accessibility:** Every team member must have access to the current version to prevent "wasted life" scenarios where developers implement canceled or outdated features.

2. Change Control Practices

The Need for Change Management

Software change is necessary because it is nearly impossible to define all requirements upfront. As development progresses, market opportunities arise, and business needs evolve. Without formal change control, an "expectation gap" develops between what stakeholders expect and what is actually delivered.

Change Control Policy

An effective organization communicates a policy stating expectations for handling changes. Key tenets include:

- **Process Adherence:** Only requests submitted through the approved process are considered.
- **No "Gold Plating":** No implementation work is performed on unapproved changes.
- **The Change Control Board (CCB):** This body decides which changes to implement; a request does not guarantee an update.
- **Transparency:** The change database must be visible to stakeholders, and the rationale for decisions must be recorded.

The Change Control Process Lifecycle

A change request typically passes through a defined lifecycle:

1. **Submission:** Received through an approved channel.
2. **Evaluation:** Assessing technical feasibility, cost, and alignment with business goals.
3. **Decision:** Approval or rejection by the CCB.
4. **Implementation:** Updating work products and tracing the change through the system.
5. **Verification:** Peer reviews or testing to ensure the change was implemented correctly.

3. Monitoring and Attributes

Requirements Status Tracking

Tracking the status of functional requirements provides a precise gauge of project progress. Rather than stating a task is "90% done," status tracking allows for reporting based on the number of requirements in specific states.

Status	Definition
Proposed	Requested by an authorized source but not yet approved.
In Progress	A business analyst is actively crafting the requirement.
Drafted	The initial version has been written.
Approved	Analyzed, impact estimated, and committed to a baseline.
Implemented	Code is designed, written, and unit tested.
Verified	Satisfied acceptance criteria and confirmed through testing.
Deferred	Approved but planned for a later release.
Rejected	Proposed but not approved for implementation.

Requirements Attributes

Attributes provide metadata for each requirement, facilitating management and filtering. Common attributes include the date created, the original source, current version, priority, status, and the rationale behind the requirement.

4. Requirements Tracing

Tracing allows project members to follow the life of a requirement both forward and backward.

Traceability Links

- **Forward Traceability:** Links customer needs (business objectives/market demands) to specific requirements, ensuring all needs are addressed.

- **Backward Traceability:** Links requirements back to their origins to ensure no unnecessary “scope creep” has occurred.
- **Downstream Tracing:** Links requirements to work products like design elements, code modules, and test cases.

The Requirements Traceability Matrix (RTM)

The RTM is a table that captures these interconnections. It helps in:

- **Impact Analysis:** Identifying all system elements affected by a proposed change.
- **Maintenance:** Locating where specific business rules are addressed in the code.
- **Testing:** Pointing to likely areas of defect when a specific test fails.

5. Glossary of Key Terms

- **Baseline:** A set of requirements agreed upon by stakeholders, typically following review and approval, which serves as the basis for further development.
- **Cardinality:** In tracing, the numerical relationship between system elements (e.g., one-to-one, one-to-many, many-to-many).
- **Change Control Board (CCB):** The body of project leaders and stakeholders that decides whether to approve or reject proposed requirement changes.
- **Change Request:** A formal proposal to modify a requirement, which must follow a defined lifecycle from submission to closure.
- **Exit Criteria:** Specific conditions that must be satisfied to indicate that a process, such as change control, has been completed successfully.
- **Expectation Gap:** The difference between what stakeholders believe is being built and the actual state of the requirements or product.
- **Impact Analysis:** The process of evaluating a proposed change to identify affected system elements and estimate the required effort and risks.
- **Requirements Management (RM) Tool:** A software application used to store requirements, track their status, manage versions, and facilitate traceability.
- **Revision History:** A record within a document or tool that identifies the changes made, the date, the person responsible, and the rationale.
- **Traceability Matrix:** A table used to represent the links between requirements and other system elements like designs, code, and tests.
- **Version Control:** The practice of uniquely identifying and tracking different iterations of a requirement or document to maintain a history of changes.