

UNIT 3

Software Requirement Analysis and Specification

(10 Hours) — Software Engineering (CACS253)

3.1 System and Software Requirements

Requirements are descriptions of the services that a system should provide and the constraints on its operation. They reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order or finding information. The process of finding out, analysing, documenting and checking these services and constraints is called **requirements engineering (RE)**.

Requirements may range from a high-level abstract statement of a service to a detailed formal specification. Hence they are written at different levels of detail for different readers:

Level	Also called	Written for	Description
User requirements	Customer requirements	Clients, managers, end-users, contractors	Statements in natural language plus diagrams of what services the system provides and constraints under which it operates.
System requirements	Software requirements	System developers, architects	A structured document giving detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented; may be part of a contract.

Example – User vs. System Requirement

User requirement: The library system shall allow members to renew borrowed books online.

System requirements: (1) The system shall allow a logged-in member to select any borrowed book and request renewal. (2) Renewal shall be refused if the book is reserved by another member. (3) The new due date shall be 14 days from renewal date and shown to the member.

3.2 Types of Software Requirements

A. Functional Requirements

Statements of services the system should provide, how the system should react to particular inputs and how it should behave in particular situations. In some cases, functional requirements may also explicitly state what the system should *not* do.

- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients expected to attend that day.
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

B. Non-Functional Requirements

Constraints on the services or functions offered by the system – such as timing, standards, and development process constraints. They often apply to the system as a whole rather than to individual features. Failing to meet a non-functional requirement can make the whole system unusable.

Category	Sub-types / Meaning	Example
Product requirements	Usability, efficiency (performance, space), dependability, security	The system shall respond to any query within 3 seconds for 95% of requests.
Organisational requirements	Environmental, operational, development process constraints	The system shall be developed using Java and comply with the company coding standard.
External requirements	Regulatory, ethical, legislative constraints (privacy, safety)	The system shall protect patient data in accordance with the national privacy law.

Metrics for specifying non-functional requirements make them testable:

Property	Measure
Speed	Processed transactions per second; user/event response time; screen refresh time
Size	Megabytes; number of ROM chips
Ease of use	Training time; number of help frames
Reliability	Mean time to failure; probability of unavailability; rate of failure occurrence; availability
Robustness	Time to restart after failure; percentage of events causing failure; probability of data corruption on failure
Portability	Percentage of target-dependent statements; number of target systems

Note: Write non-functional requirements quantitatively wherever possible. “The system shall be user-friendly” is untestable; “A novice user shall be productive after 2 hours of training” is testable.

C. Domain Requirements

Requirements that come from the **application domain** of the system and reflect characteristics and constraints of that domain. They may be functional or non-functional and are usually expressed in the language of the domain, which software engineers may not understand well. This can lead to problems: the requirements may be so obvious to experts that they are never stated (implicit knowledge).

Example – Domain Requirement

A library system: “Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will be either printed locally on the system server for manually forwarding to the user or routed to a network printer.”

D. User Requirements

User requirements should describe functional and non-functional requirements so they are understandable by system users who do not have detailed technical knowledge. Write using natural language, simple tables, forms and intuitive diagrams. Avoid technical jargon and implementation detail.

- Lack of clarity – precision is difficult without making the document hard to read.
- Requirements confusion – functional, non-functional requirements and design goals not clearly distinguished.
- Requirements amalgamation – several different requirements may be expressed together.

Functional Requirement	Non-Functional Requirement
Describes <i>what</i> the system does	Describes <i>how well</i> the system performs it
Specific feature or function (login, generate bill)	System-wide quality (speed, security, usability)
Usually implemented in specific modules	Affects overall architecture
Verified by functional testing	Verified by performance, security and usability testing

3.3 Requirements Engineering Process

Requirements Engineering Process

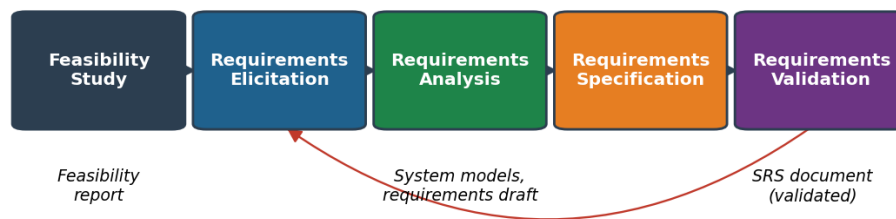


Figure 3.1: Requirements engineering process (activities iterate)

The requirements engineering process includes **feasibility study, elicitation and analysis, specification and validation**. In practice these activities are interleaved and iterative, and requirements management (handling change) runs throughout.

3.4 Elicitation and Analysis of Requirements

Requirements elicitation and analysis is the process of deriving system requirements through observation of existing systems, discussion with potential users and procurers, task analysis and so on. Software engineers work with customers and system end-users to find out about the application domain, what services the system should provide, the required performance, hardware constraints, and so on.

Process activities:

- **Requirements discovery** – interacting with stakeholders to discover their requirements; domain requirements are also discovered.
- **Requirements classification and organisation** – grouping related requirements and organising them into coherent clusters.
- **Requirements prioritisation and negotiation** – prioritising requirements and resolving requirements conflicts.
- **Requirements specification** – requirements are documented and input into the next round of the spiral.

Problems of requirements analysis: stakeholders do not know what they really want; they express requirements in their own terms; different stakeholders have conflicting requirements; organisational and political factors influence requirements; and requirements change during the analysis process as new stakeholders emerge and the business environment evolves.

Overview of Techniques

Technique	Description
Interviews	Direct question-and-answer sessions with stakeholders.
Questionnaires / surveys	Structured questions to a large number of users.
Observation (ethnography)	Watching users perform their real work.
Scenarios and use cases	Describing sequences of interaction with the system.
Document analysis	Studying existing forms, reports, manuals and old systems.
Prototyping	Showing a working sample to draw out feedback.
Joint application development (JAD) / workshops	Structured group sessions with all stakeholders.

Viewpoints

Different stakeholders see the system differently. **Viewpoint-oriented elicitation** recognises these perspectives and uses them to structure and organise both the elicitation process and the requirements themselves. A viewpoint is a way of looking at the problem that reflects the interest of a group of stakeholders.

Type of Viewpoint	Description	Example (ATM system)
Interactor viewpoints	People or other systems that interact directly with the system	Bank customers, bank staff
Indirect viewpoints	Stakeholders who do not use the system but influence requirements	Bank managers, security staff
Domain viewpoints	Domain characteristics and constraints that influence requirements	Banking standards, communication protocols

Interviewing

Formal or informal interviews with stakeholders are a part of most RE processes. Two types exist:

- **Closed interviews** – based on a pre-determined list of questions.
- **Open interviews** – various issues are explored with stakeholders, without fixed agenda.

In practice a mix of both is used. Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system, but they are not good for understanding domain requirements (jargon is used and some knowledge is so familiar it is difficult to articulate) and organisational requirements.

- **Be open-minded**, avoid pre-conceived ideas about the requirements and be willing to listen.
- **Prompt the interviewee** with a spring-board question, a requirements proposal or by working together on a prototype system.

Scenarios

Scenarios are real-life examples of how a system can be used. They help elicit requirements because people find it easier to relate to real-life examples than abstract descriptions. A scenario should include: a description of the starting situation; a description of the normal flow of events; a description of what can go wrong; information about other concurrent activities; and a description of the state when the scenario finishes.

Scenario – Withdraw Cash (ATM)

Initial assumption: The user has a card and knows PIN.

Normal: User inserts card → enters PIN → selects Withdraw and amount → takes card → takes cash.

What can go wrong: Wrong PIN entered three times → card retained; insufficient balance → request declined.

Other activities: Bank may be doing end-of-day processing.

System state on completion: Balance updated, transaction logged, card returned.

Use Cases

Use cases are a scenario-based technique in the UML that identify the **actors** in an interaction and describe the interaction itself. A set of use cases should describe all possible interactions with the system. Actors are represented by stick figures and each use case by an ellipse; lines link actors to use cases. High-level graphical models are supplemented by more detailed tabular descriptions.

Use Case Element	Content
Use case name	Issue Book
Actors	Librarian (primary), Member (secondary)
Pre-condition	Member is registered; book is available
Main flow	1. Librarian scans member card. 2. Scans book. 3. System checks limit and availability. 4. System records loan and due date.
Alternative flow	Member has overdue books → issue refused with message.

Use Case Element	Content
Post-condition	Loan recorded; book status = 'issued'

*Note: Use-case relationships: **include** (mandatory reuse of another use case) and **extend** (optional/conditional behaviour).*

Ethnography

Ethnography is an observational technique that can be used to understand social and organisational requirements. An analyst immerses himself or herself in the working environment where the system will be used, observes day-to-day work and notes the actual tasks in which participants are involved. The value of ethnography is that it helps discover implicit system requirements that reflect the actual rather than the formal processes in which people are involved. People often find it very difficult to describe their work, as it is second nature to them.

- Effective for discovering two types of requirements: requirements derived from the way in which people actually work rather than how process definitions say they ought to work; and requirements derived from cooperation and awareness of other people's activities.
- Not suitable for discovering new features to be added to a system; combined with prototyping for best effect.

3.5 Requirements Validation

Requirements validation is concerned with showing that the requirements actually define the system that the customer wants. Requirements error costs are high, so validation is very important – fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Validation Check	Question
Validity checks	Does the system provide the functions which best support the customer's needs?
Consistency checks	Are there any requirements conflicts?
Completeness checks	Are all functions required by the customer included?
Realism checks	Can the requirements be implemented given available budget and technology?
Verifiability	Can the requirements be checked (testable)?

Validation techniques: requirements reviews (systematic manual analysis by a team of client and contractor staff), prototyping (executable model to check requirements) and test-case generation (developing tests for requirements; if a test is difficult or impossible to design, the

requirement is usually hard to implement).

3.6 Requirements Specification

Requirements specification is the process of writing down the user and system requirements in a requirements document. The document should be a clear, unambiguous statement for everybody involved – customers, managers, engineers and testers – and should state *what* the system should do rather than *how*.

Notation	Description
Natural language sentences	Numbered sentences expressing a requirement; universal and understandable but ambiguous.
Structured natural language	Standard forms or templates limit the freedom of the writer and ensure uniformity.
Graphical notations	UML use-case, sequence and data-flow diagrams supplement text.
Mathematical (formal) specifications	Based on mathematical concepts such as finite-state machines or sets; unambiguous, but few customers understand them.

Structure of a Software Requirements Specification (IEEE 830)

- **1. Introduction** – purpose, scope, definitions, references, overview.
- **2. Overall description** – product perspective, functions, user characteristics, constraints, assumptions.
- **3. Specific requirements** – functional requirements, external interface requirements, performance, design constraints, quality attributes.
- **4. Appendices and index.**

Characteristics of a good SRS: correct, unambiguous, complete, consistent, ranked for importance, verifiable, modifiable and traceable.

3.7 Feasibility Study

A feasibility study decides whether or not the proposed system is worthwhile. It is a short, focused study that checks: does the system contribute to organisational objectives; can the system be engineered using current technology and within budget; and can the system be integrated with other systems that are used?

Feasibility Type	Question Answered
Technical	Is the required technology available and are the skills present?
Economic	Do the benefits outweigh the costs? (cost-benefit analysis, ROI, payback period)
Operational	Will the system be used and accepted by the users and fit the organisation's operation?
Schedule	Can the system be completed in the available time?
Legal	Does it comply with laws, regulations and contracts?

Example – Economic Feasibility

A system costs Rs. 10,00,000 to develop and Rs. 1,00,000 per year to run, and saves Rs. 4,00,000 per year.

Net annual benefit = 4,00,000 – 1,00,000 = Rs. 3,00,000. **Payback period** = $10,00,000 \div 3,00,000 \approx 3.3$ years.

Key Takeaways

- ✓ Requirements are user (customer-level, natural language) or system (detailed) statements of services and constraints.
- ✓ Functional = what the system does; Non-functional = product, organisational and external constraints; Domain = from the application area.
- ✓ Elicitation techniques: interviews, viewpoints, scenarios, use cases, ethnography, prototyping, questionnaires.
- ✓ Use cases identify actors and interactions; scenarios describe concrete flows including what can go wrong.
- ✓ Validation checks validity, consistency, completeness, realism and verifiability – errors are very expensive if found late.
- ✓ The SRS states *what* not *how*; follow IEEE 830 structure.
- ✓ Feasibility study covers technical, economic, operational, schedule and legal aspects.

— End of Unit 3 —