REQUIREMENT ENGINEERING

Soon Phei Tin | 2025

WHAT WE'LL ACHIEVE TODAY

- Apply RE processes to a case study
- Perform elicitation, analysis, specification, and validation
- Reflect on challenges like ambiguities and conflicts

SOFTWARE PROCESSES

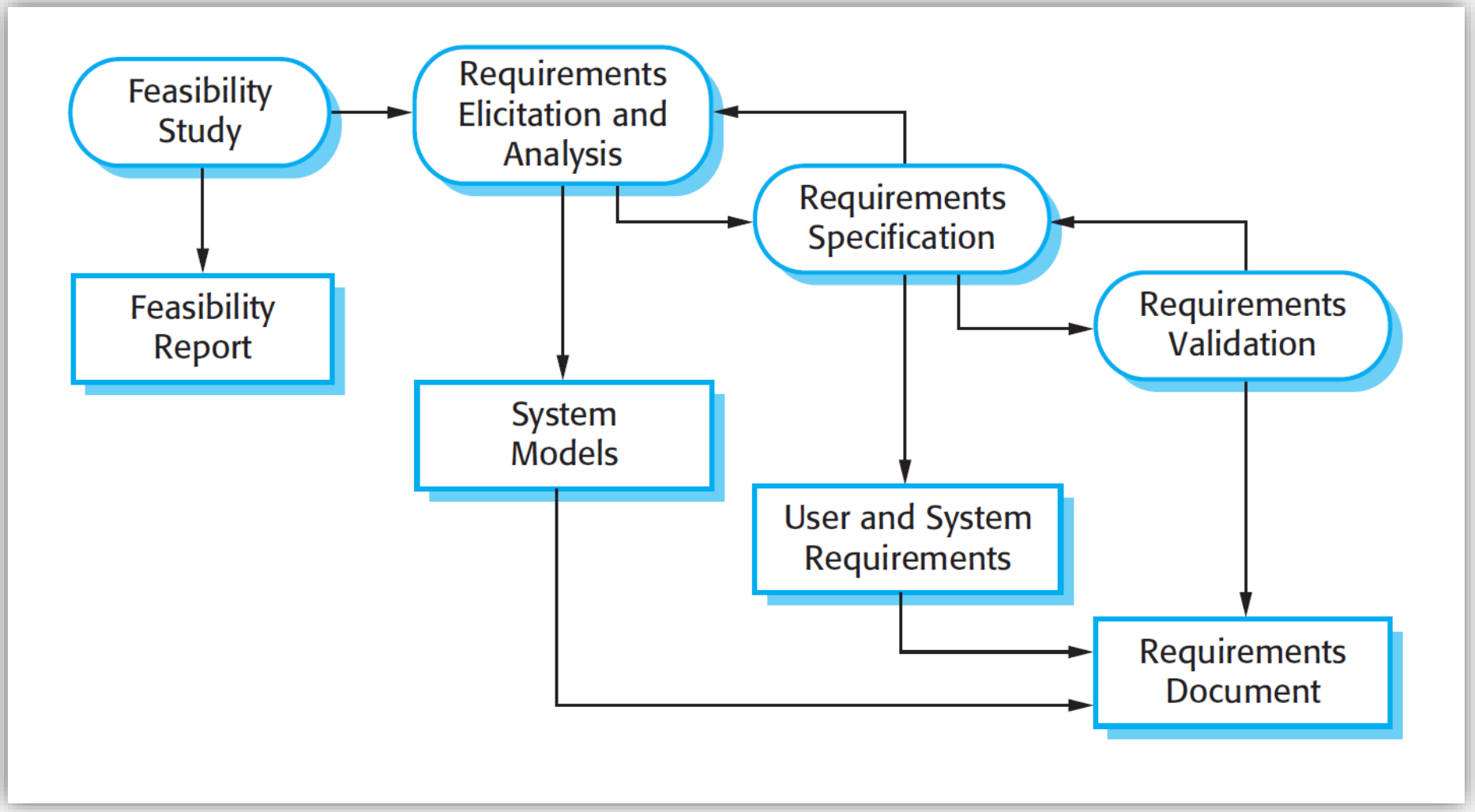
- FOUR fundamental activities

WHAT IS REQUIREMENT ENGINEERING?

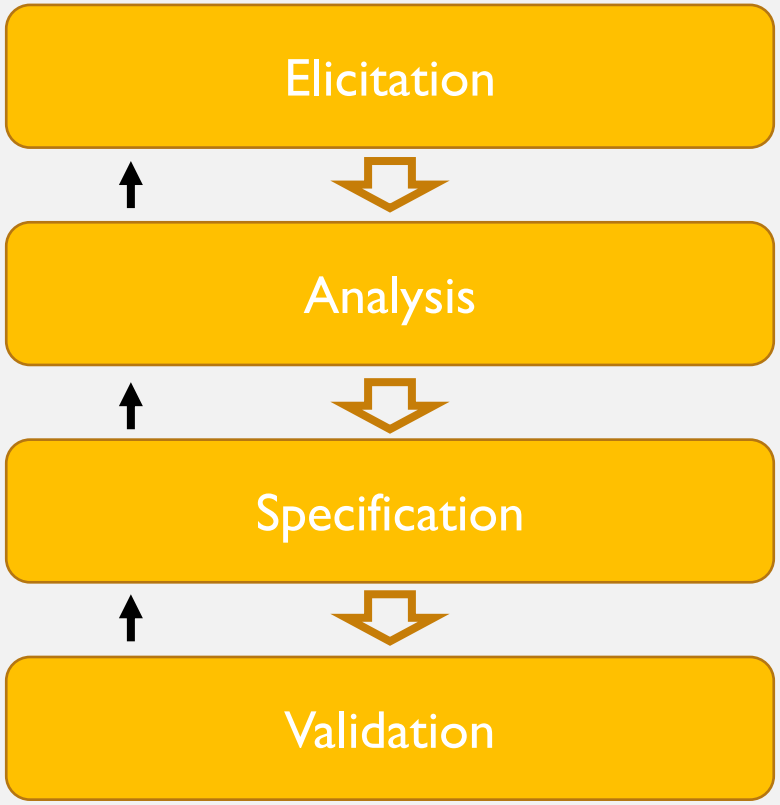
- The requirements for a system are the descriptions of what the system should do.

- The process of finding out, analyzing, documenting and checking these needs and constraints is called requirements engineering (RE).

OLD SCHOOL


Feasibility Study
Requirements Elicitation and Analysis
Requirements Specification
Requirements Validation
Feasibility Report
System Models
User and System Requirements
Requirements Document

RE ROADMAP



CASE STUDY: UNIVERSITY TIMETABLING APP

- Build an app for scheduling classes, rooms, and alerts. Stakeholders: Students, lecturers, admins.
- We will complete the RE for the case study following the RE Roadmap.

STEP 1: ELICITATION

- Goal: Gather requirements from stakeholders via various techniques. e.g. interviews and scenarios
- Software engineers work with stakeholders to find out about: -
  - the application domain
  - what services the system should provide
  - the required performance of the system
  - hardware constraints, and so on

STEP 1: ELICITATION

- Stakeholders could be: -
  - end users who will interact with the system
  - anyone else in an organization who will be affected by it
  - engineers who are developing or maintaining other related systems
  - business managers
  - domain experts
  - trade union representatives

STEP 1: ELICITATION

- Sources of information during the elicitation phase include documentation, system stakeholders, and specifications of similar systems.
  - interviews
  - observation
  - scenario
  - prototypes

THE INTERVIEW

THE INTERVIEW

SCENARIO

- User can understand and criticize a scenario of how they might interact with a software system.
- Use the information gained from the discussion based on a scenario to formulate the actual system requirements.
- A scenario is the descriptions of example interaction sessions. Each scenario usually covers one or a small number of possible interactions
- A scenario starts with an outline of the interaction. During the elicitation process, details are added to this to create a complete description of that interaction.

SCENARIO 1: STUDENT VIEWS SCHEDULE

- **Actor:** Sarah (student).
- **Context:** Sarah opens the app on her iPhone to check her weekly classes. (based on the interview)
- **Steps:**
  - Sarah logs in with her university ID.
  - The app displays her timetable for the week (e.g., Mon: CS301, Room A-12, 9 AM).
  - She taps a class to see details (lecturer, room capacity, notes).
- **Exceptions:** Login fails (wrong ID).

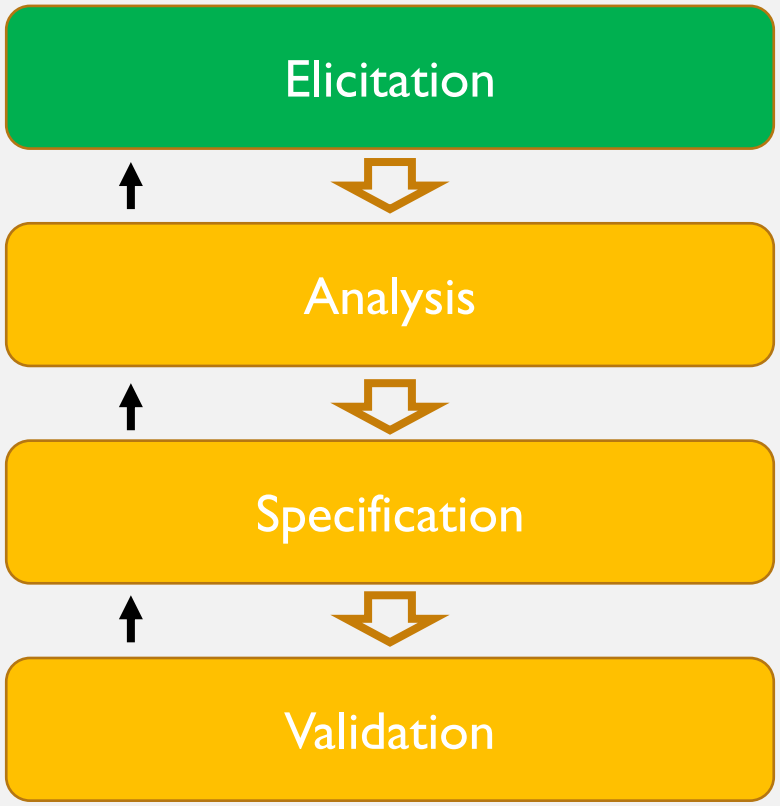
SCENARIO 2: LECTURER REQUESTS ROOM CHANGE

- **Actor:** Dr. Li (lecturer).
- **Context:** Dr. Li finds his lecture room double-booked.
- **Steps:**
  - Dr. Li logs into the web app.
  - He views his schedule and selects the conflicting slot (e.g., Wed 2 PM).
  - He requests a new room from a list of available rooms.
  - The system sends a confirmation email after admin approval.
- **Exceptions:** No rooms available; admin rejects request.

FINALIZE THE ELICTATION

- Refine these scenarios: Add details from interviews (e.g., specific timings, UI preferences).
- Conduct additional interviews if missing information is identified.
- You may need to repeat the process until the requirements are complete and satisfactory.

RE ROADMAP



1. Interview
2. Scenario

STEP 2: ANALYSIS

- **Goal:** Organize requirements, resolve conflicts, and prioritize

- **Organize requirements:** Group requirements into functional (services) and non-functional (constraints).
- **Resolve conflicts:** Identify conflicts
- **Prioritize:** Use MoSCoW (Must/Should/Could/Won't) to prioritize

FUNCTIONAL / NON-FUNCTIONAL

- Functional requirements: These are statements of services the system should provide. How the services should react and behave in certain condition.

- Non-functional requirements: These are constraints on the services or functions offered by the system. Non-functional requirements often apply to the system as a whole, rather than individual system features or services.

FUNCTIONAL / NON-FUNCTIONAL

- Requirements are extracted from stakeholder interviews (Student, Lecturer, Admin) and scenarios. They are classified as:
  - **Functional Requirements**: Define services or behaviors the system must provide (e.g., specific features or actions).
  - **Non-Functional Requirements**: Define constraints or qualities (e.g., performance, security, accessibility).

FUNCTIONAL REQUIREMENTS

ID	Requirement	Source	Notes
FR1	The system shall allow students to view their weekly timetable on mobile devices.	Sarah (Interview), Scenario 1	Includes class details (e.g., room, lecturer).
FR2	The system shall allow lecturers to request a room change for conflicting slots.	Dr. Li (Interview), Scenario 2	Requires availability check and approval.
FR3	The system shall generate monthly room usage reports for administrators in PDF format.	Ms. Lee (Interview), Scenario 3	Includes occupancy rates.
FR4	The system shall send push notifications to students for schedule changes or cancellations.	Sarah (Interview), Scenario 1	Triggered by updates.
FR5	The system shall send email notifications to lecturers for room change confirmations.	Dr. Li (Interview), Scenario 2	After admin approval.

FUNCTIONAL REQUIREMENTS

- The functional requirements specification of a system should be both complete and consistent.
  - Completeness means that all services required by the user should be defined.
  - Consistency means that requirements should not have contradictory definitions.
- In practice, for large, complex systems, it is practically impossible to achieve requirements consistency and completeness.
  - it is easy to make mistakes and omissions when writing specifications for complex systems
  - there are many stakeholders in a large system. Stakeholders have different and often inconsistent needs.

NON-FUNCTIONAL REQUIREMENTS

ID	Requirement	Source	Notes
NFR1	The system shall load timetables within 3 seconds on mobile devices.	Sarah (Interview, clarified "quick")	Performance constraint.
NFR2	The system shall comply with WCAG 2.1 accessibility standards, including text-to-speech support.	Sarah, Ms. Lee (Interview)	Legal and inclusivity requirement.
NFR3	The system shall achieve 99.9% uptime for reliability.	Ms. Lee (Interview, implied "reliable")	Critical for user trust.
NFR4	The system shall support both iOS and Android mobile platforms.	Sarah (Interview)	Cross-platform compatibility.
NFR5	The system shall ensure secure login to prevent unauthorized access.	Dr. Li (Interview)	Security requirement.

NON-FUNCTIONAL REQUIREMENTS

- Non-functional requirements, as the name suggests, are requirements that are not directly concerned with the specific services delivered by the system to its users.
- System properties such as reliability, response time, and store occupancy.
- Constraints on the system implementation such as performance, security, or availability.
- Usually specify constrain characteristics of the system as a whole
- Often more critical than individual functional requirements

NON-FUNCTIONAL REQUIREMENTS

- Often more critical than individual functional requirements
- failing to meet a non-functional requirement can mean that the whole system is unusable
- The implementation of non-functional requirements may be intricately dispersed throughout the system.
  - They may affect the overall architecture of a system rather than the individual components.
  - A single non-functional requirement may generate a number of related functional requirements.

NON-FUNCTIONAL REQUIREMENTS

- A common problem with non-functional requirements is that users or customers often propose these requirements as general goals, such as …

The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized.

vs

Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use.

CONFLICTS

- Conflict: **Performance vs. Security**
- NFR 1 - The system shall load timetables within 3 seconds on mobile devices.
- NFR 6 - The system shall ensure secure login to prevent unauthorized access.
- Justification: Adding robust security (e.g., multi-factor authentication) may increase load time beyond 3 seconds, especially on low-end devices.
- Resolution: Use lightweight OAuth 2.0 for login, balancing speed and security. Test to ensure <3s load on average devices.

CONFLICTS

- Conflict: **Accessibility vs. Development Time**
- NFR 2: The system shall comply with WCAG 2.1 accessibility standards, including text-to-speech support.
- Justification: Full WCAG compliance (e.g., text-to-speech) requires extensive UI testing, which may delay release.
- Resolution: Focus on core WCAG features (e.g., high-contrast UI, screen reader support) in first release; defer advanced features like text-to-speech to later iterations.

CONFLICTS

- Conflict: **Room Change Speed vs. Approval**
- FR 2: The system shall allow lecturers to request a room change for conflicting slots.
- FR 6: The system shall allow administrators to approve or reject room change requests.
- Justification: Lecturers want immediate room changes (Dr. Li), but admin approval (Ms. Lee) adds delay.
- Resolution: Allow temporary room swaps pending approval; notify admins for fast review.

MOSCOW PRIORITIZATION TECHNIQUE

- **MoSCoW** is a prioritization method used to rank requirements based on their importance to the project's success. It helps balance stakeholder needs against constraints (e.g., budget, timeline). The acronym stands for:
  - **Must Have**: Critical requirements without which the system is unusable or fails to meet core objectives (e.g., core functionality, legal compliance).
  - **Should Have**: Important but not critical; can be deferred if necessary (e.g., enhances user experience).
  - **Could Have**: Desirable but lower priority; implemented if time/resources allow (e.g., nice-to-have features).
  - **Won't Have**: Out of scope for this release, possibly considered later (e.g., futuristic features).

EXERCISE

- FR1: View weekly timetable on mobile
- Priority: Must
- Justification: Core functionality for students; primary user need (Sarah).

EXERCISE

- FR 2: Request room change
- Priority: Should
- Justification: Important for lecturers but less frequent than viewing (Dr. Li).

EXERCISE

- FR 3: Generate monthly room usage reports
- Priority: Should
- Justification: Key for admins but not daily use (Ms. Lee)

EXERCISE

- FR 8: Authenticate users via university ID
- Priority: Must
- Justification: Essential for security and access control (Scenario 1)

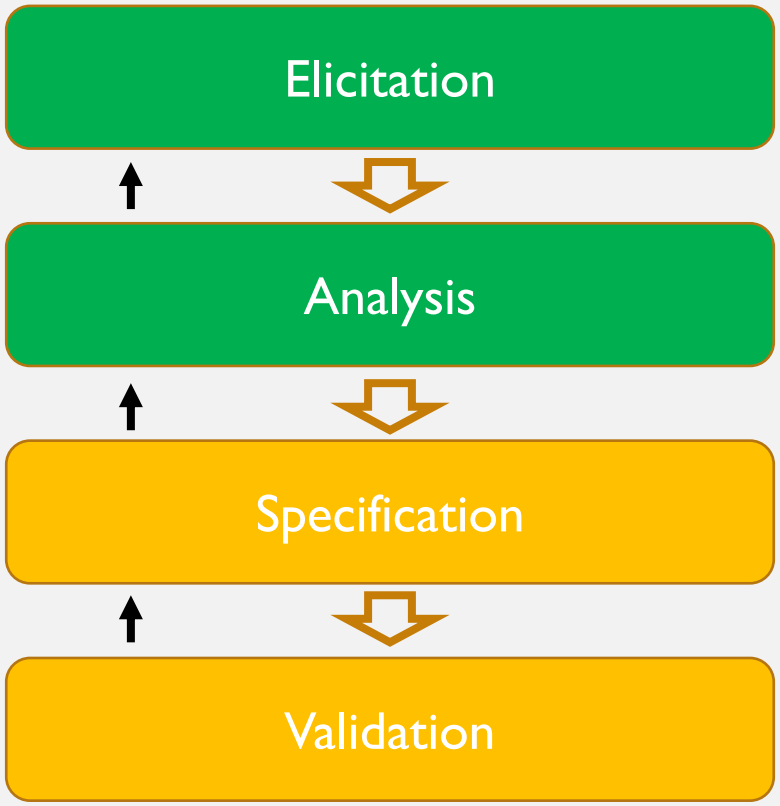
EXERCISE

- NFR 1: Load timetables within 3 seconds
- Priority: Should
- Justification: Improves UX but not critical if slightly slower (Sarah)

PRIORITIZED REQUIREMENTS

ID	Requirement	Priority	Justification
FR1	View weekly timetable on mobile	Must	Core functionality for students; primary user need (Sarah)
FR2	Request room change	Should	Important for lecturers but less frequent than viewing (Dr. Li)
…	…	…	…
NFR1	Load timetables within 3 seconds	Should	Improves UX but not critical if slightly slower (Sarah).
NFR2	Comply with WCAG 2.1	Must	Legal requirement; ensures inclusivity (Sarah, Ms. Lee)
…	…	…	…

RE ROADMAP



1. Interview
2. Scenario

1. Organize Requirements
2. Identify & resolve conflicts
3. Prioritize Requirements

USER REQUIREMENTS VS SYSTEM REQUIREMENTS

- High-level user requirements, which focus on what the system should do from the stakeholders' perspective.
- These requirements are abstract, user-oriented, and serve as a bridge between stakeholder needs and system design.
- Low-level system requirements, on the other hand, dive into the technical details of how the system will implement those needs.
- Low-level system requirements should be written after high-level user requirements are validated and approved.

STEP 3: SPECIFICATION

- The process of writing down the user and system requirements in a requirements document.
- The user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent.
- Stakeholders interpret the requirements in different ways and there are often inherent conflicts and inconsistencies in the requirements.

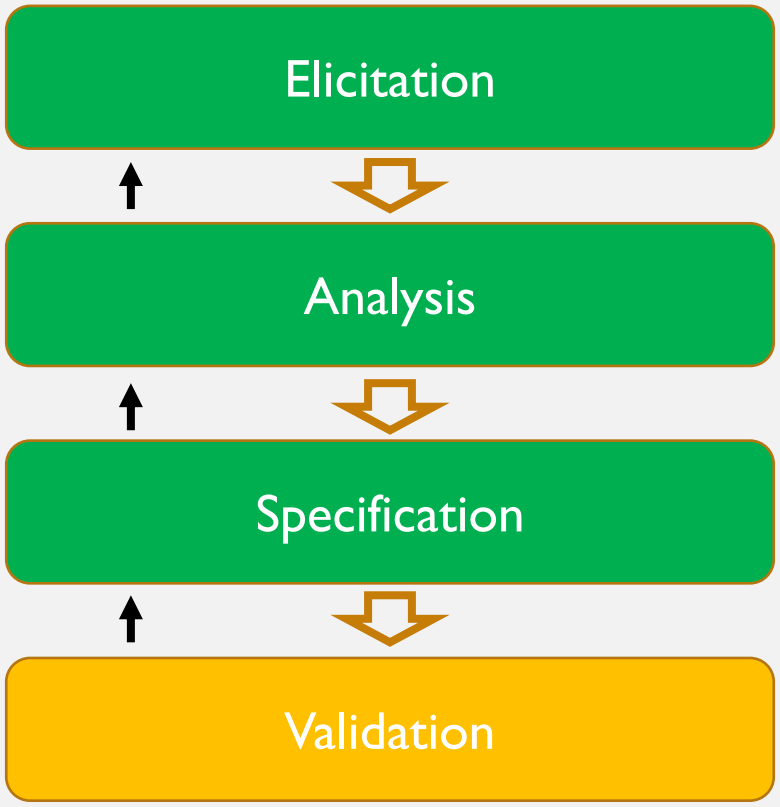
STEP 3: SPECIFICATION

- The user requirements for a system should describe the functional and nonfunctional requirements in a way they are understandable by system users who don't have detailed technical knowledge
- Specify only the external behavior of the system.
- Should not include details of the system architecture or design
- Write user requirements in natural language, with simple tables, forms, and intuitive diagrams.

STEP 3: SPECIFICATION

- Refer to the sample SRS

RE ROADMAP



1. Interview
2. Scenario

1. Organize Requirements
2. Identify & resolve conflicts
3. Prioritize Requirements

1. SRS

STEP 4: VALIDATION

- Requirements validation is the process of checking that requirements correctly define the system that the customer really wants
- It is important because errors in a requirements document can lead to extensive rework costs when these problems are discovered during development or after the system is in service.

STEP 4: VALIDATION

- During the requirements validation process, different types of checks should be carried out on the requirements in the requirements document. These checks include:
  - *Validity checks* - The functions proposed by stakeholders should be aligned with what the system needs to perform. You may find later that there are additional or different functions are required instead.
  - *Consistency checks* - Requirements in the document should not conflict. That is, there should not be contradictory constraints or different descriptions of the same system function.

STEP 4: VALIDATION

- *Completeness checks* - The requirements document should include requirements that define all functions and the constraints intended by the system user.
- *Realism checks* - Using knowledge of existing technology, the requirements should be checked to ensure that they can be implemented realistically.
- *Verifiability* - To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable.

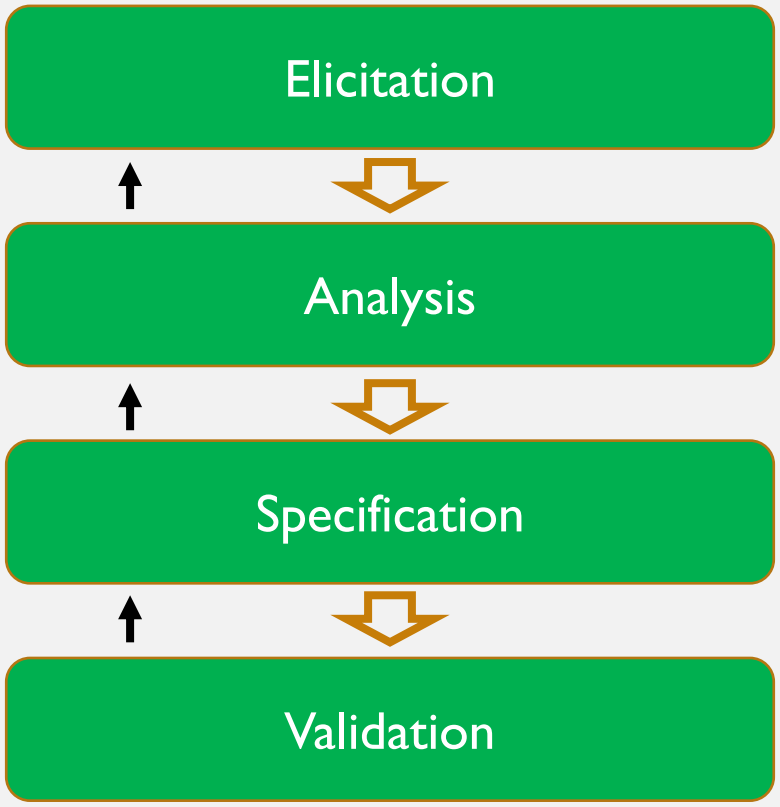
STEP 4: VALIDATION

- There are a number of requirements validation techniques that can be used individually or in conjunction with one another:
  - *Requirements reviews* - The requirements are analyzed systematically by a team of reviewers who check for errors and inconsistencies.
  - *Prototyping* - In this approach to validation, an executable model of the system in question is demonstrated to end-users and customers
  - *Test-case generation* - Requirements should be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems. If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered

VALIDATION EXAMPLES

- Refer to the PDF

RE ROADMAP



1. Interview
2. Scenario

1. Organize Requirements
2. Identify & resolve conflicts
3. Prioritize Requirements

1. SRS

1. Refined SRS

CHALLENGES

- Stakeholder Conflicts
  - Different stakeholders often have conflicting priorities, making consensus difficult.
  - Delays in agreement can stall the process
- Ambiguities in Requirements
  - Vague or unclear terms (e.g., “efficiently,” “timely”) lead to misinterpretation by developers or stakeholders
  - Ambiguities can cause rework

CHALLENGES

- Incomplete or Evolving Requirements
  - Requirements may be missing or change over time as stakeholders gain clarity.
  - Incomplete requirements risk functionality gaps, while evolving needs require iterative RE, challenging fixed timelines
- Resource Constraints
  - Limited time, budget, or personnel restrict the depth of elicitation, analysis, and validation.
  - Constraints mirror real-world startups, potentially compromising quality if not managed.

CHALLENGES

- Validation Difficulties
  - Ensuring requirements are testable, realistic, and meet all scenarios is challenging, especially for non-functional requirements
  - Validation failures, can lead to critical errors if not thoroughly addressed
- Communication Gaps
  - Misalignment between technical teams and non-technical stakeholders (e.g., admins vs. developers) can lead to misunderstandings.
  - Poor communication, common in large-scale projects (e.g., Boeing 787 delays), can lead to requirements misalignment.

END