



Chapter Three

Requirement Elicitation and Analysis

Compiled by: | Sinodos G

Dire Dawa - 2017 E.C

Introduction

- ▶ The process to gather the software requirements from client, analyze and document them is known as requirement Engineering.
- ▶ Any project typically starts with a set of requirements
- ▶ For any project [a short-term or a long-term], it is very important that the requirements are structured, clear and complete
- ▶ A complete, clear and structured requirements are ensured by requirement elicitation and analysis



Requirement Analysis



Requirement Elicitation
and Analysis

- ▶ Involves refining, prioritizing, and validating requirements to ensure feasibility and clarity.
- ▶ It involves requirement elicitation, projecting out all the possible alternate and error flows in the proposed solution
- ▶ Typically, development of user scenarios, identifying use cases, prototypes can be considered for the analysis.



Requirement Analysis Techniques

- Refining and Validating Requirements
- Ensure the gathered requirements are clear, complete, feasible, and free of conflicts.

► *Techniques*

- Use Case & User Stories
- Prototyping
- Document Analysis
- Requirement Workshops
- Mind Mapping
- Feasibility Study
- Prioritization Techniques

Requirement Elicitation



Requirement Elicitation
and Analysis

- ▶ The process of gathering information from stakeholders to understand what they need in a system or project.
- ▶ Requirement elicitation is the process of bringing out all the details required for the successful implementation of a solution altering to the demand of the end-users and other stakeholders
- ▶ Requirement elicitation can be broadly classified as functional and non-functional.
- ▶ Requirement Elicitation is Asking "What do you need?"

Discussion

- ▶ **Examples:** Imagine you are developing a mobile app for online food delivery.
 - Identify list of system users
 - Discuss how would gather requirements from users
 - List some of the basic requirements
 - Show how unclear or missing requirements could lead to problems

- - - - // - - - -

- | | |
|---|--|
| <ul style="list-style-type: none">▶ Identify list of system users<ul style="list-style-type: none">• Customers• Restaurant owners, and• Delivery person | <ul style="list-style-type: none">▶ Gathering Requirements from Users<ul style="list-style-type: none">• Customers (End Users)• Restaurant Owners• Delivery Personnel |
|---|--|

Example - basic requirements of mobile app

▶ **For Customers**

- User-friendly interface for browsing restaurants and menus
- Multiple payment options (credit card, PayPal, cash on delivery)
- Order tracking with estimated delivery time

▶ **For Restaurant Owners**

- Dashboard to manage orders and update menu items
- Integration with inventory systems
- Performance reports (sales, customer feedback, delivery times)

▶ **For Delivery Personnel**

- GPS navigation for route optimization
- Order status updates (e.g., "Picked Up," "Out for Delivery")
- Secure payment tracking system

Requirement Elicitation vs Requirement Analysis

Purpose

- Collecting raw requirements
- Refining and validating requirements

Focus

- Understanding needs
- Ensuring feasibility, clarity, and consistency

Common Methods

- Interviews, surveys, focus groups, observations
- Prototyping, feasibility studies, prioritization, mind mapping

Example Activity

- Asking students how they prefer to mark attendance
- Checking if the attendance method is secure and practical

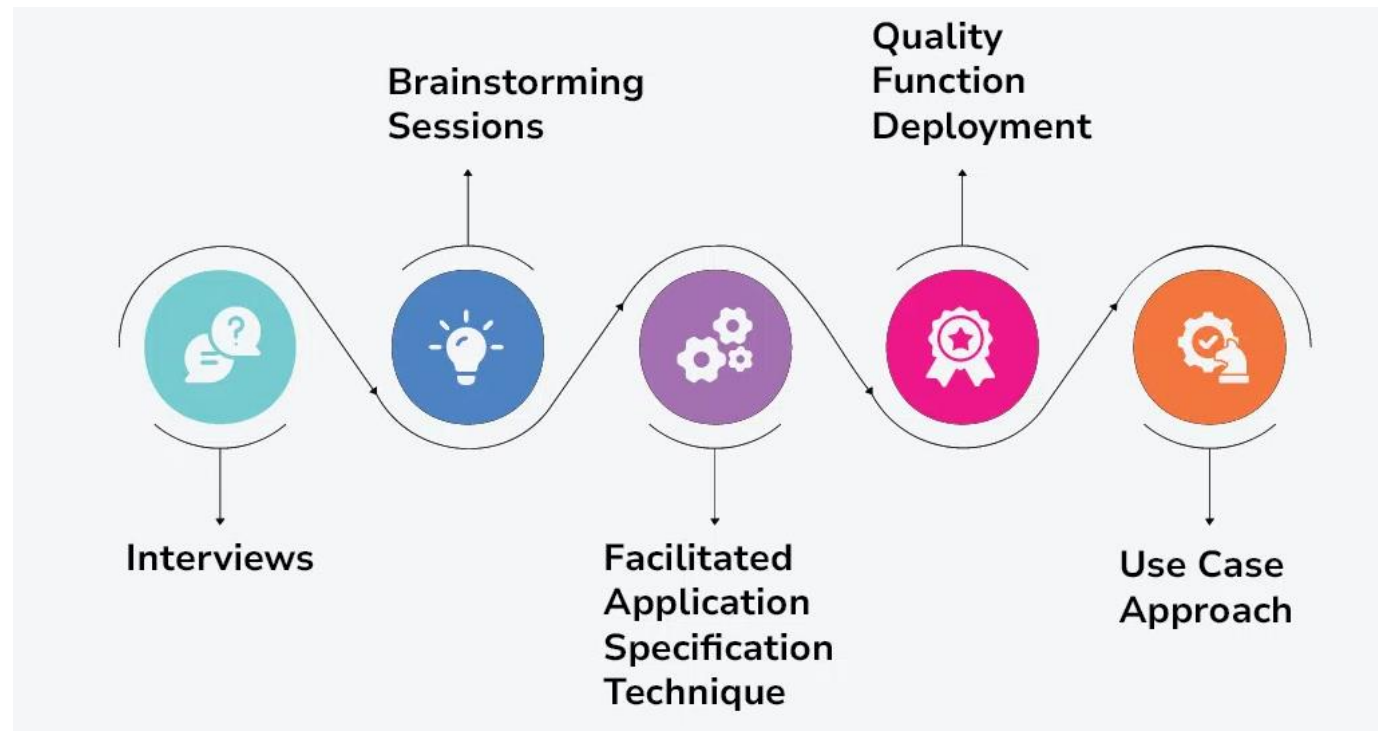
Elicitation Techniques

- ▶ Specific techniques which may be used to collect knowledge about system requirements
- ▶ This knowledge must be structured
 - Partitioning - aggregating related knowledge
 - Abstraction - recognising generalities
 - Projection - organising according to perspective
- ▶ Elicitation problems
 - Not enough time for elicitation
 - Inadequate preparation by engineers
 - Stakeholders are unconvinced of the need for a new system



Cont'd . . .


- ▶ Collect raw requirements from stakeholders through direct or indirect methods.
- ▶ There are several requirements elicitation methods.
- ▶ List of common Techniques
 - Interviews
 - Surveys
 - Questionnaires
 - Observation
 - Brainstorming
 - Document Analysis
 - Focus Groups
 - Prototyping
 - Use Cases & User Stories



Interviews

- It is impossible to interview every stakeholder hence, representatives from groups are selected based on their expertise and credibility.
- Interviews may be open-ended or structured.
- In open-ended interviews, there is no pre-set agenda. Context-free questions may be asked to understand the problem.
- In a structured interview, an agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

Interview Steps

- 
- ▶ Prepare
 - ▶ Conduct
 - Opening
 - Body
 - Closing
 - ▶ Follow through

► **Brainstorming**

- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

► **Surveys**

- Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

▶ **Questionnaires**

- A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.
- A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

▶ **Task analysis**

- Team of engineers and developers may analyze the operation for which the new system is required.
- If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

▶ **Domain Analysis**

- Every software falls into some domain category.
- The expert people in the domain can be a great help to analyze general and specific requirements.

▶ **Brainstorming**

- An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.
- Conclusions which aid to form requirements expected from the software.

► ***Prototyping***

- used to classify criteria which are incomplete or undefined.
- by designing the prototypes, regular demonstrations are offered to the consumer so that consumers can get an idea of how the product would look.
- building user interface without adding detail functionality for user to interpret the features of intended software product.
- The prototype is shown to the client and the feedback is noted.

► **Observation**

- experts visit the clients organization or workplace.
- Used to observe the actual working of the existing installed systems.
- The team itself draws some conclusions which aid to form requirements expected from the software.

► ***Laddering***

- A structural type of dialogue in which a small number of standard questions are addressed to stakeholders.
- The question series is structured in a hierarchical order, and the performance of this approach depends on the understanding of the domain stakeholders.

Quality Function Deployment

- ▶ Refer to the customer satisfaction of expected system development.
- ▶ There are three types of requirements are identified:-
 - ▶ **Normal requirements**
 - the objective and goals of the proposed software are discussed with the customer.
 - **Example:** normal requirements for a result management system may be entry of marks, calculation of results, etc.
 - ▶ **Expected requirements**
 - These requirements are so obvious that the customer need not explicitly state them.
 - **Example:** protection from unauthorized access.
 - ▶ **Exciting requirements**
 - features that are beyond customer's expectations and prove to be very satisfying when present. **Example:** when unauthorized access is detected, it should back up and shut down all processes.

Use Case Approach

- ▶ Use Case technique combines text and pictures to provide a better understanding of the requirements.
- ▶ The use cases describe the ‘what’, of a system and not ‘how’.
- ▶ They only give a functional view of the system.
- ▶ The components of the use case design include three major things
 - Actor, use cases, and use case diagram.
- ▶ **Actor**
 - ▶ the external agent that lies outside the system but interacts with it
 - ▶ may be a person, machine, etc.
 - ▶ It is represented as a stick figure.
 - ▶ Actors can be primary actors or secondary actors.
 - ▶ Primary actors: It requires assistance from the system to achieve a goal.
 - ▶ Secondary actor: It is an actor from which the system needs assistance.

▶ ***Use cases:***

- They describe the sequence of interactions between actors and the system.
- They capture who(actors) do what(interaction) with the system.
- A complete set of use cases specifies all possible ways to use the system.

▶ ***Use case diagram***

- ▶ A use case diagram graphically represents what happens when an actor interacts with a system.
- ▶ It captures the functional aspect of the system.
 - A stick figure is used to represent an actor.
 - An oval is used to represent a use case.
 - A line is used to represent a relationship between an actor and a use case.

Software Requirements

- ▶ Requirement elicitation can be broadly classified into **Functional** and **Non-Functional** requirements.
- ▶ However, modern software requirements also include
 - *Functional Requirements*
 - *Non-Functional Requirements*
 - *Domain Requirements,*
 - *Inverse Requirements, and*
 - *Design & Implementation Constraints*



Functional Requirements (FRs)

- ▶ These define the core functions and features of the system.
- ▶ They describe **what** the system should do.

Examples:

- User authentication (login, logout, password reset).
- Recording student attendance in an attendance management system.
- Generating reports for administrators.
- Allowing teachers to mark attendance manually or through facial recognition.

Non-Functional Requirements (NFRs)

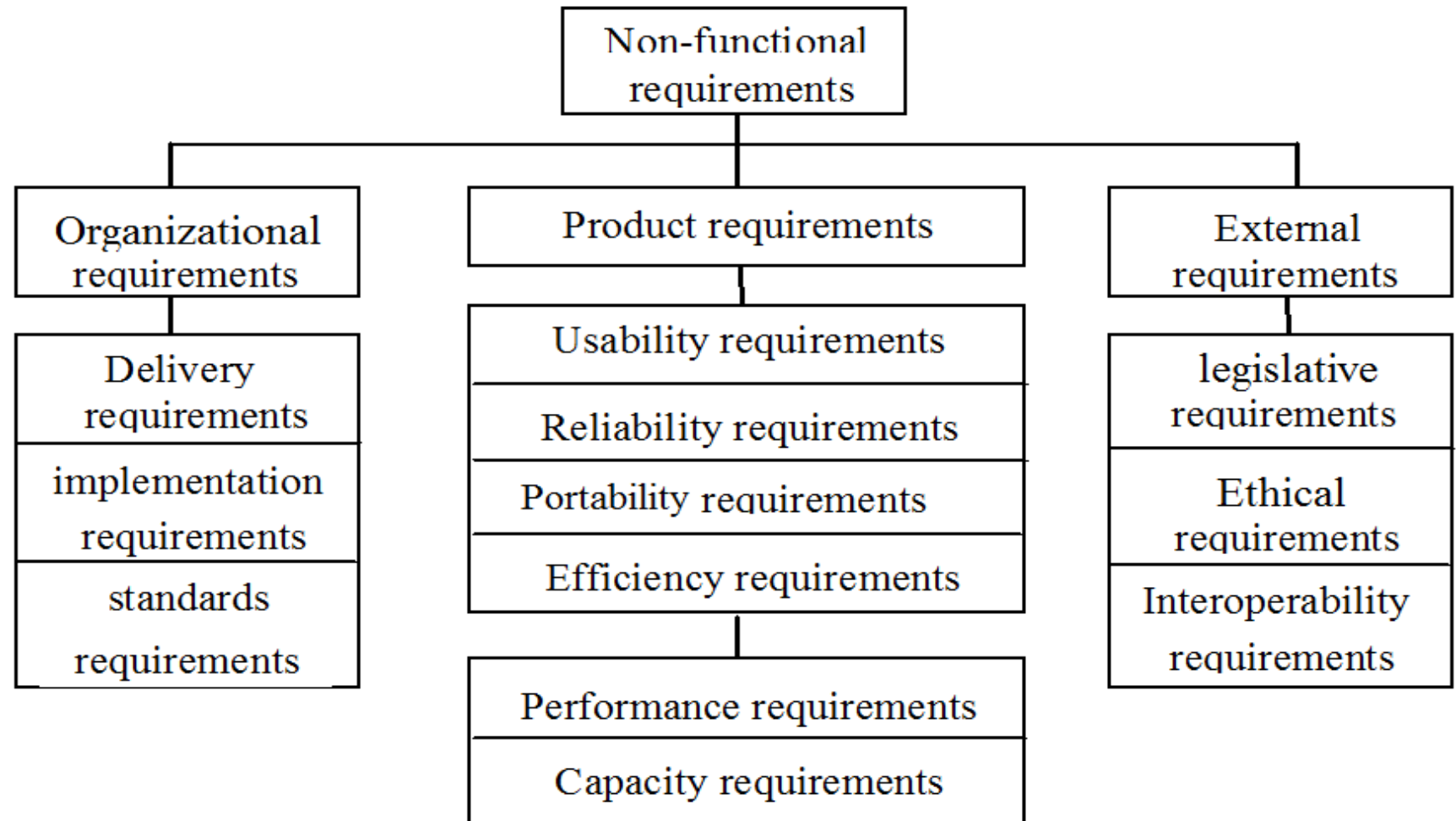
- ▶ These define the quality attributes and performance standards of the system.
- ▶ They describe **how** the system should function.
- ▶ define the overall qualities or attributes of the resulting system
- ▶ place restrictions on the product being developed, the development process, and specify external constraints that the product must meet.

Examples:

- **Performance:** The system should process attendance records within 2 seconds.
- **Security:** All user data must be encrypted using AES-256.
- **Scalability:** The system should support up to 10,000 concurrent users.
- **Usability:** The user interface should be simple and intuitive.

Classification of NFRs

- NFRs may be classified in terms of qualities that a software must exhibit
- A more general classification is
 - product,
 - Organizational and
 - External requirements



Product requirements

- ▶ Specify the desired characteristics that a system or subsystem must possess.
- ▶ Most NFRs are concerned with specifying constraints on the behaviour of the executing system.
- ▶ Product
 - Usability, reliability, Portability, efficiency (performance, space)
- ▶ *Example:*
 - The system shall allow one hundred thousand hits per minute on the website.
 - The shall not have down time of more than one second for continues execution of the 1000 hours.



Organizational Requirement

- ▶ Derived from the policies and procedures of the customers or the developing organizations.
- ▶ Organizational
 - Standards, implementation, delivery
- ▶ ***Example:***
 - The system development process and deliverable documents shall confirm to the MIL-STD-2167A
 - Any development work sub – contracted by the development organization shall be carried out in accordance with capability maturity model.



External Requirement

- ▶ Which are derives from the factors, external to the system, external to the software system and sometimes organization and its development process.
- ▶ External
 - Interoperability, ethical, legislative (privacy, safety)
- ▶ Example:
 - The system shall not disclose any personal information about members of the library system to other members, except system administrator



NFRs as Goals

- ▶ Non-functional requirements are sometimes written as general goals, which are difficult to verify
- ▶ They should be expressed quantitatively using metrics (measures) that can be objectively tested

Example:

- ▶ Goal (unverifiable)
 - The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized.
- ▶ Goal (***verifiable***)
 - Experienced controllers shall be able to use all the system functions after a total of two hours' training.
 - After this training, the average number of errors made by experienced users shall not exceed two per day

Metrics for Non-Functional Requirements

Metrics for NFRs - 1

<i>Property</i>	<i>Measure</i>
Speed	<ol style="list-style-type: none">1. Processed transactions/second2. Response time3. Screen refresh time
Reliability	<ol style="list-style-type: none">1. Mean time to failure2. Probability of unavailability3. Rate of failure occurrence4. Availability

Metrics for NFRs - 2

<i>Property</i>	<i>Measure</i>
Size	<ol style="list-style-type: none">1. K bytes2. Number of function points
Robustness	<ol style="list-style-type: none">1. Time to restart after failure2. Percentage of events causing failure3. Probability of data corruption on failure
Ease of use	<ol style="list-style-type: none">1. Training time2. Number of help frames

Cont'd . . .

<i>Property</i>	<i>Measure</i>
Robustness	<ol style="list-style-type: none">1. Time to restart after failure2. Percentage of events causing failure3. Probability of data corruption on failure
Portability	<ol style="list-style-type: none">1. Percentage of target-dependent statements2. Number of target systems

Domain Requirement

- ▶ reflect the environment in which the system operates so, when we talk about an application domain we mean environments such as train operation, medical records, e-commerce etc.
- ▶ These are industry-specific or organizational constraints that influence system design.
- ▶ They may include regulatory standards, business rules, or specialized data formats.
- ▶ Important because they often reflect fundamentals of the application domain.
- ▶ If these requirements are not satisfied, it may be impossible to make the system work satisfactorily.



▶ ***Examples:***

- ▶ In a healthcare system, patient records must comply with HIPAA regulations.
- ▶ A banking application must support two-factor authentication.
- ▶ An e-learning platform must allow LMS integration (e.g., SCORM or xAPI support).
- ▶ The hospital management system should support the check-in and check out for IPD (In Patient department) and OPD (Out patient Department)
- ▶ System will keep claim records & losses reserves, Financial Transactions and Claim expense records
- ▶ Supporting associate agents at the policy level setup commission due based on written premium, on invoicing or on premium payments and generating agent statements

Inverse Requirement

- ▶ It describes the constraints on allowable behavior.
- ▶ In many cases, it is easier to state that certain behavior must never occur than to state requirements guaranteeing acceptable behavior in all circumstances.
- ▶ These define what the system should not do.
- ▶ They help prevent unintended functionalities.
- ▶ Examples:



Examples of Inverse Requirements

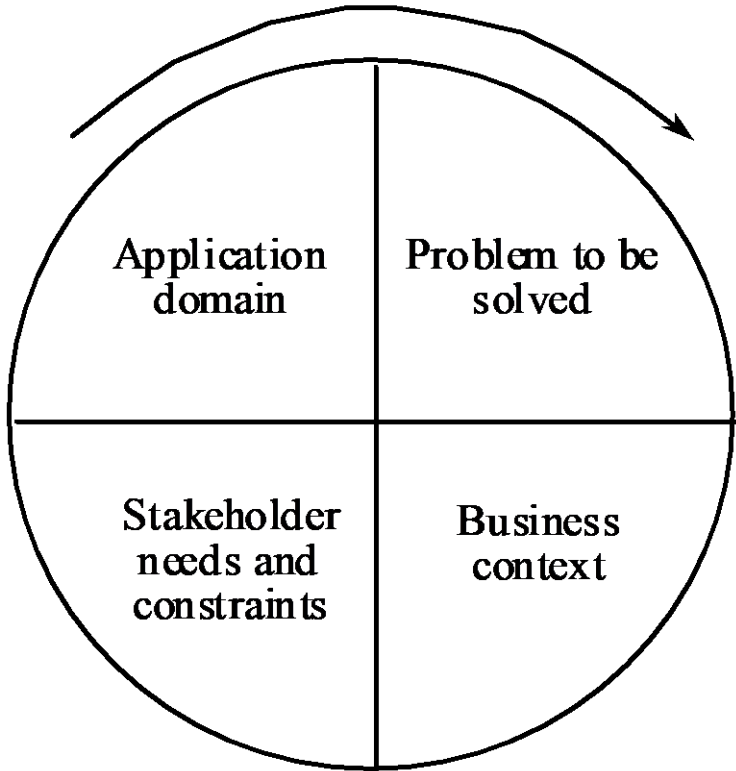
- ▶ User ID should only contain digits.
- ▶ The system should not use red color in the user interface, whenever it asking for a input from the user.
- ▶ The system must not allow students to modify their own attendance records.
- ▶ The application should not store passwords in plaintext.
- ▶ A financial system must not process transactions without authentication.



Design and Implementation Constrains

- ▶ boundary conditions on how the software is to be constructed and implemented.
- ▶ These are technical and environmental limitations that affect system design and development.
- ▶ They are givens of the development within which the designer must work.
- ▶ **Examples:**
 - ▶ Platform Constraint: The application must run on Windows, Linux, and macOS.
 - ▶ Technology Constraint
 - The system must be developed using PHP (CodeIgniter 3) and MySQL.
 - ▶ Hardware Constraint
 - The system must support fingerprint scanners for attendance tracking.
 - ▶ Budget Constraint
 - The total project cost must not exceed \$50,000.

Components of requirements elicitation



▶ **Application domain understanding**

- Application domain knowledge is knowledge of the general area where the system is applied.

▶ **Problem understanding**

- The details of the specific customer problem where the system will be applied must be understood.

▶ **Business understanding**

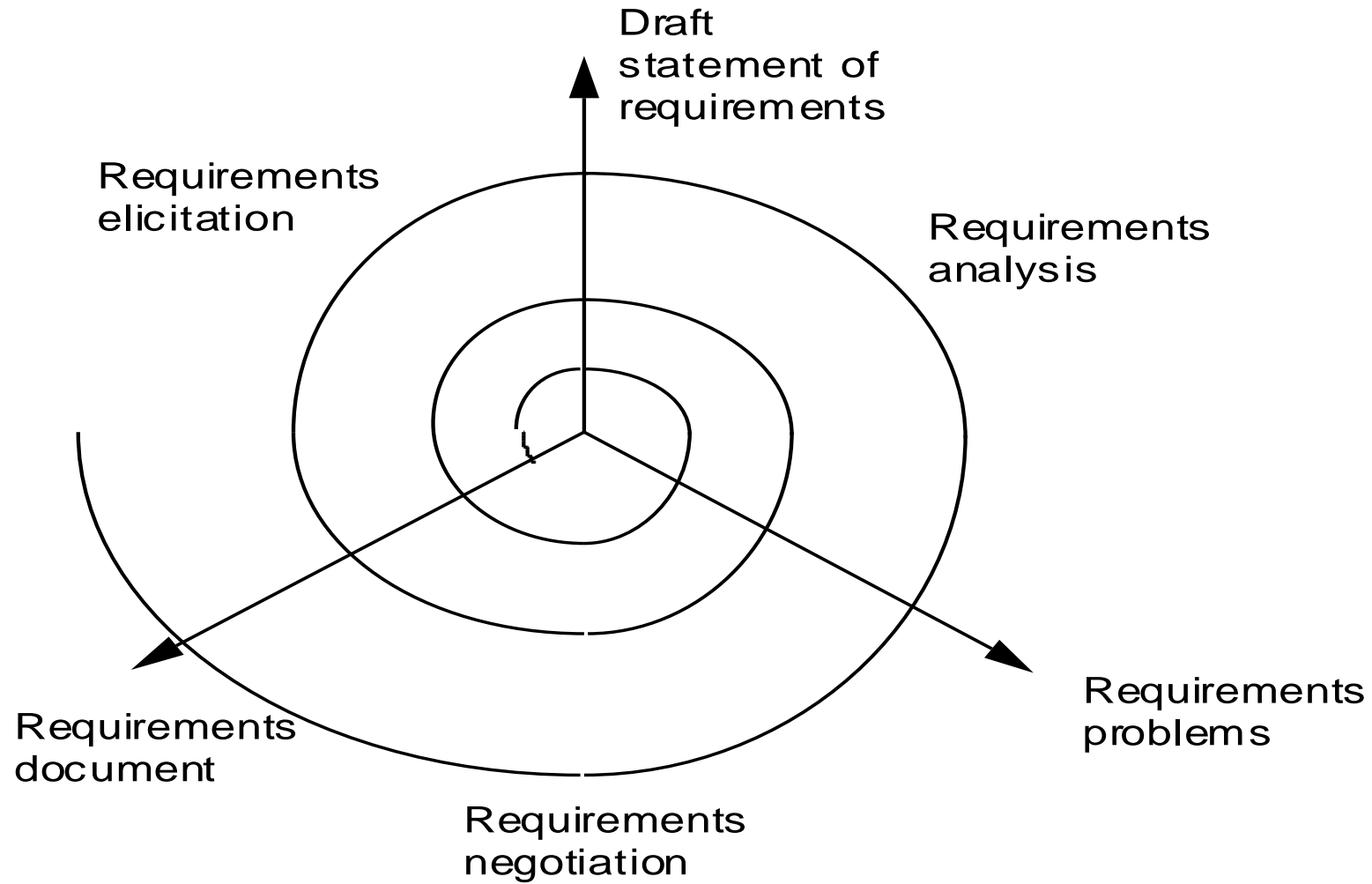
- You must understand how systems interact and contribute to overall business goals.

▶ **Constraints of system stakeholders**

- You must understand, in detail, the specific needs of people who require system support in their work.



Elicitation, analysis and negotiation

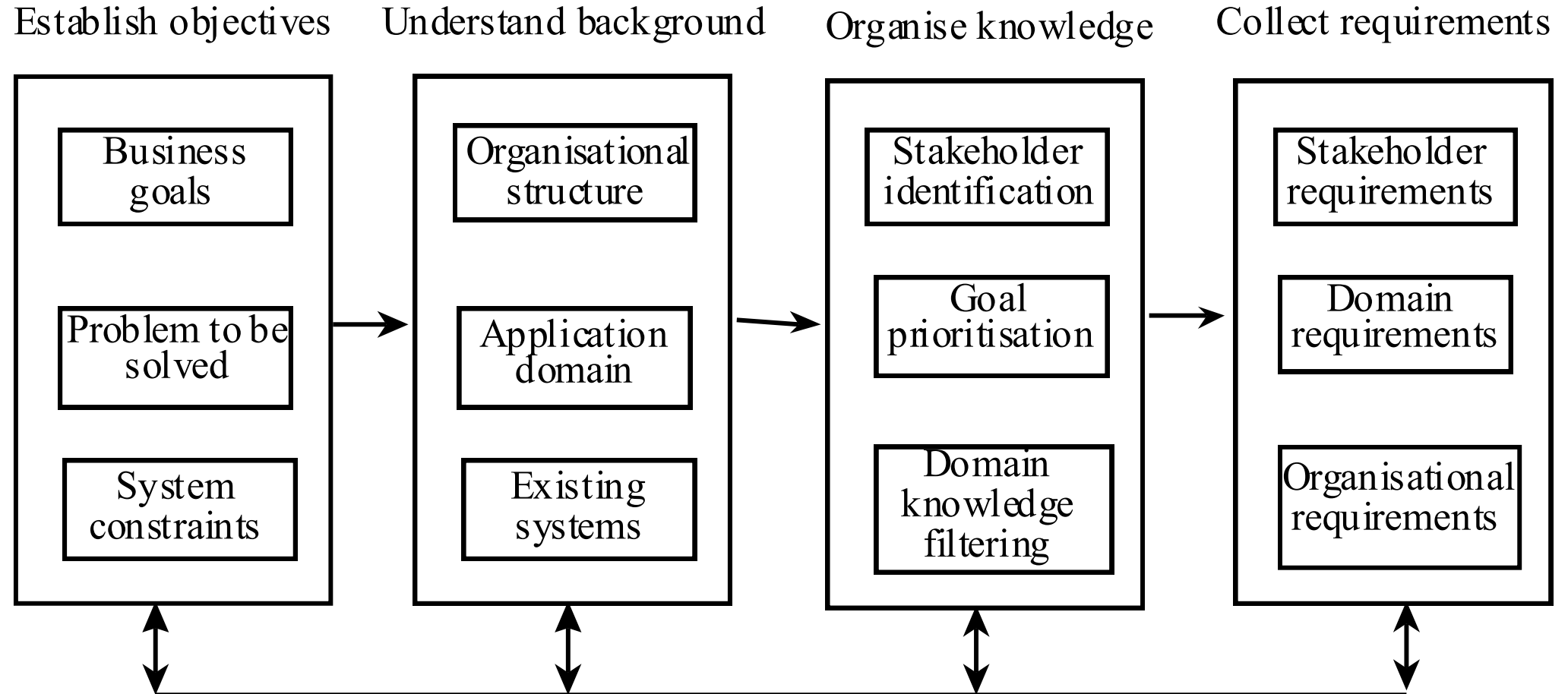


Elicitation stages

- ▶ Objective Setting
 - The organisational objectives should be established including general goals of the business, an outline description of the problem to be solved, why the system is necessary and the constraints on the system.
- ▶ Background knowledge acquisition
 - Background information about the system includes information about the organisation where the system is to be installed, the application domain of the system and information about existing systems
- ▶ Knowledge organisation
 - The large amount of knowledge which has been collected in the previous stage must be organised and collated.
- ▶ Stakeholder requirements collection
 - System stakeholders are consulted to discover their requirements.



The requirements elicitation process



Requirement Elicitation Process



- ▶ There are four Requirement Elicitation Process
 - Requirements gathering
 - Organizing Requirements
 - Negotiation & discussion
 - Documentation

▶ ***Requirements gathering***

- The developers discuss with the client and end users and know their expectations from the software.

▶ ***Organizing Requirements***

- The developers prioritize and arrange the requirements in order of importance, urgency and convenience.

▶ ***Negotiation & discussion***

- To remove the ambiguity and conflicts, they are discussed for clarity and correctness.
- Unrealistic requirements are compromised reasonably.

▶ ***Documentation***

- All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Software Requirements Characteristics

- ▶ Gathering software requirements is the foundation of the entire software development project.
 - Clear, Correct
 - Consistent
 - Coherent
 - Comprehensible
 - Modifiable
 - Verifiable
 - Prioritized
 - Unambiguous
 - Traceable
 - Credible source

Requirements Reuse

- ▶ Reuse involves taking the requirements which have been developed for one system and using them in a different system
- ▶ The practice of leveraging existing requirements
 - such as functional specifications, constraints, and design elements from previous projects or systems to improve efficiency, consistency, and quality in new or similar projects.
- ▶ Requirements reuse saves time and effort as reused requirements have already been analysed and validated in other systems
- ▶ Currently, requirements reuse is an informal process but more systematic reuse could lead to larger cost savings



► **Types of Requirements Reuse:**

- **Full Reuse** – Using entire sets of requirements without modifications (e.g., standard system functions).
- **Partial Reuse** – Adapting and modifying existing requirements for a new context.
- **Pattern-Based Reuse** – Using requirement templates or best practices from previous projects.

► **Benefits of Requirements Reuse:**

- **Time and Cost Savings** – Reduces effort in gathering and documenting new requirements.
- **Improved Consistency** – Ensures uniformity across projects by using well-defined and validated requirements.
- **Higher Quality** – Reused requirements are often well-tested, reducing errors and ambiguities.
- **Faster Development** – Accelerates system design and implementation by minimizing redundant work.
- **Knowledge Retention** – Captures and maintains institutional knowledge over time.

Reuse Possibilities

▶ Reusing Across Similar Projects

- If an organization develops multiple products with similar functionalities (e.g., banking software, e-commerce platforms), existing requirements can be reused with minor adjustments.

▶ Component-Based Reuse

- Software systems often have modular components with well-defined requirements. These components can be reused across different projects.
- Example: Authentication modules, payment gateways, and reporting systems often have standardized requirements that can be reused.

▶ Reuse in Agile and Iterative Development

- Agile teams can reuse user stories, acceptance criteria, and backlog items across sprints or projects.
- Example: A "Reset Password" user story from one project can be applied to another with minor modifications.

Challenges in requirements reuse

- Context Mismatch: Requirements may not fully align with new project needs.
- Outdated Information: Old requirements may be obsolete or irrelevant.
- Legal and Compliance Issues: Regulations may differ across projects.
- Dependency on Legacy Systems: Constraints from past implementations may limit innovation.
- Context Dependency: Requirements may not fully fit a new project without adaptation.
- Technological Changes: Outdated requirements might need modifications due to new advancements.
- Lack of Documentation: Poorly written requirements reduce the effectiveness of reuse.

Prototyping

- ▶ The process of creating an early model of a system to visualize, test, and refine requirements before full-scale development.
- ▶ It helps in gathering feedback, reducing misunderstandings, and improving final system quality.
- ▶ A prototype is an initial version of a system which may be used for experimentation
- ▶ valuable for requirements elicitation because users can experiment with the system and point out its strengths and weaknesses.
- ▶ Rapid development of prototypes is essential so that they are available early in the elicitation process



Types of Prototyping

▶ **Throwaway (Rapid) Prototyping**

- A temporary model used to refine requirements, then discarded.
- **Example:** A simple wireframe for a mobile app interface.

▶ **Evolutionary Prototyping**

- A continuously improved prototype that eventually becomes the final product.
- **Example:** A web application that starts with basic features and expands over iterations.

▶ **Incremental Prototyping**

- ▶ Different system modules are prototyped separately and later combined.
- ▶ **Example:** Developing a payment gateway prototype separately from a shopping cart system.

▶ **Extreme Prototyping** (*for Web Applications*)

- ▶ A three-step approach involving:
 - ▶ Static HTML pages (UI design)
 - ▶ Functional screen simulations
 - ▶ Fully functional backend integration

► Benefits of Prototyping

- Improves Requirement Clarity – Helps stakeholders visualize system functionality.
- Reduces Development Risks – Detects issues early in the process.
- Enhances User Involvement – Encourages feedback and iterative improvements.
- Saves Costs and Time – Prevents costly changes in later development stages.
- Supports Better Decision-Making – Provides insights before full implementation.

Prototyping costs and problems

- ▶ **Training costs** - prototype development may require the use of special purpose tools
- ▶ **Development costs** - depend on the type of prototype being developed
- ▶ **Extended development schedules** - developing a prototype may extend the schedule although the prototyping time may be recovered because rework is avoided
- ▶ **Incompleteness** - it may not be possible to prototype critical system requirements
- ▶ **Misleading Expectations** – Users may assume the prototype is the final system.
- ▶ **Resource Intensive** – Requires extra time and effort in early stages.
- ▶ **Poorly Documented Requirements** – Over-reliance on prototypes may lead to lack of formal documentation.



Approaches to prototyping

- ▶ Paper prototyping
 - a paper mock-up of the system is developed and used for system experiments
- ▶ ‘Wizard of Oz’ prototyping
 - a person simulates the responses of the system in response to some user inputs
- ▶ Executable prototyping
 - Called Automated prototyping



► Paper prototyping

- Is a cheap and surprisingly effective approach to prototype development.
- It is most suitable for establishing end-user requirements for software systems.
- Paper versions of the screens which might be presented to the end-user are drawn and various usage scenarios are planned.
- Analysts and end-users work through these scenarios to find users reactions to the system, the information they require and how they would normally interact with the system.

► ***Wizard of Oz' prototyping***

- Is also relatively cheap as it does not require much software to be developed.
- The user interacts with what appears to be the system but his or her inputs are actually channeled to a person who simulates the responses of the system.
- This approach is particularly useful when a new system has to be developed based on an existing interface.
- Users are familiar with the interface and can see the interactions between it and the system functionality simulated by the 'Wizard of Oz'.

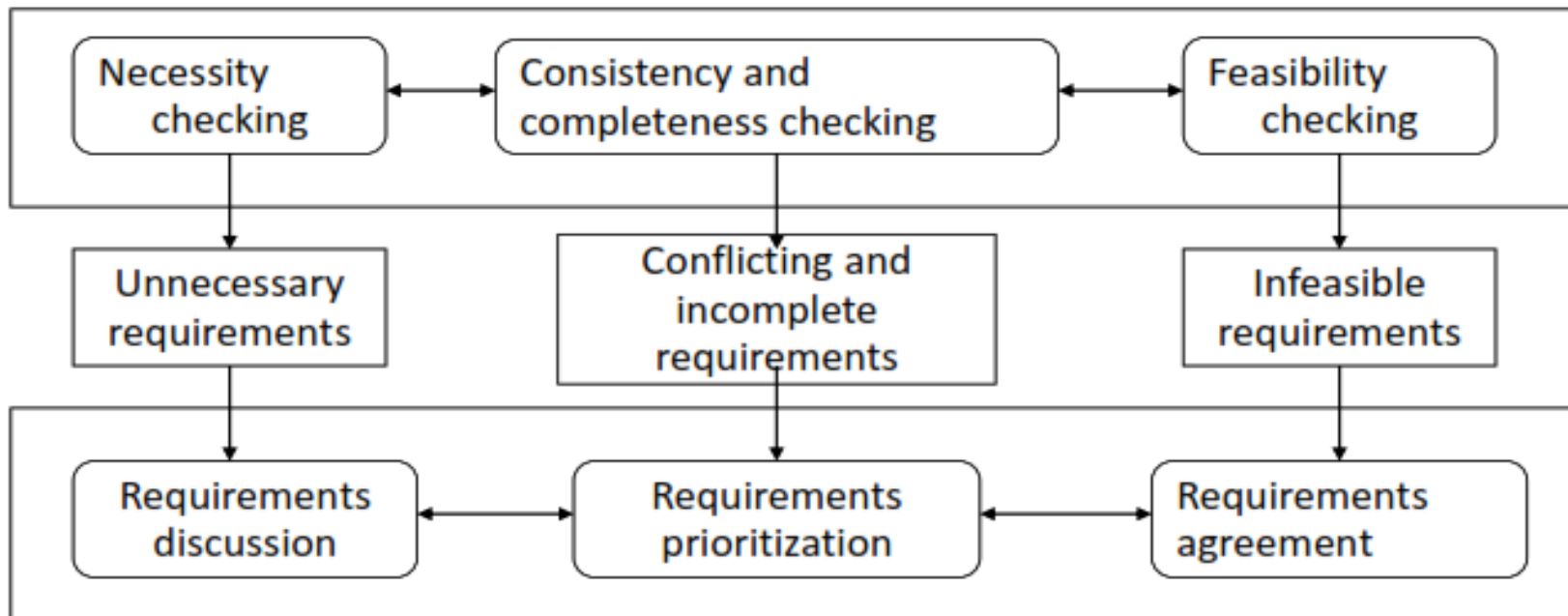
► Executable prototyping

- Called Automated prototyping
- a fourth generation language or other rapid development environment is used to develop an executable prototype
- Fourth generation languages based around database systems
- Visual programming languages such as Visual Basic or Object Works
- Internet-based prototyping solutions based on World Wide Web browsers and languages such as Java

Requirements Analysis and Negotiation

- Requirements analysis and negotiation are inter-leaved activities and join to form a major activity of the requirements engineering process.

Requirements Analysis

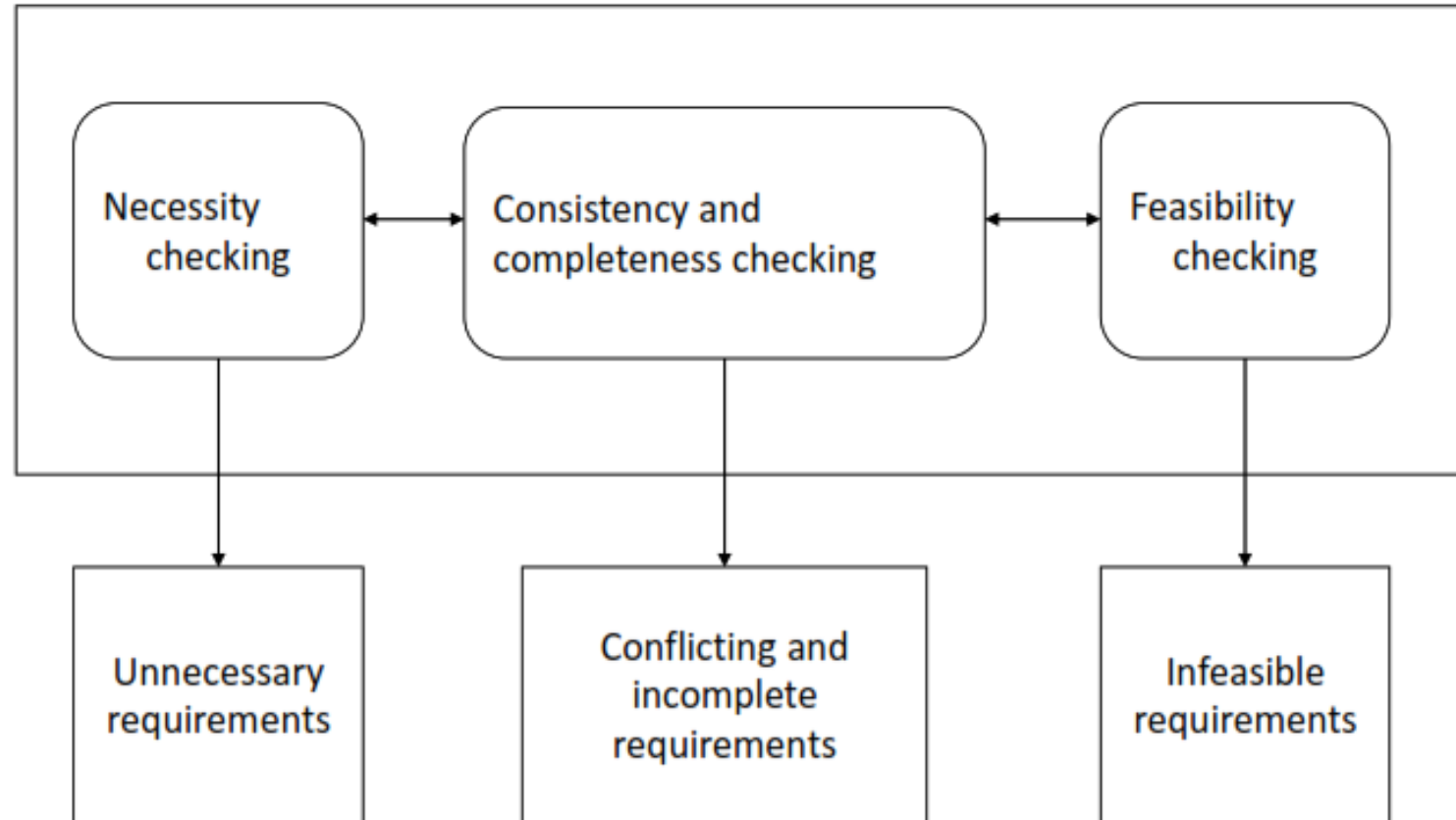


Requirements Negotiation

20

Requirements Analysis Process

Requirements Analysis



26

Requirements Interactions

- ▶ A very important objective of requirements analysis is to discover the interactions between requirements and to highlight requirements conflicts and overlaps.
- ▶ A requirements interaction matrix shows how requirements interact with each other, which can be constructed using a spreadsheet.
- ▶ Each requirement is compared with other requirements, and the matrix is filled as follows:
 - ▶ For requirements which conflict, fill in a 1
 - ▶ For requirements which overlap, fill in a 1000
 - ▶ For requirements which are independent, fill in a 0
 - ▶ Consider the following example

- ▶ Consider the following example
- ▶ An Interaction Matrix

Requirement	R1	R2	R3	R4	R5	R6
R1	0	0	1000	0	1	1
R2	0	0	0	0	0	0
R3	1000	0	0	1000	0	1000
R4	0	0	1000	0	1	1
R5	1	0	0	1	0	0
R6	1	0	1000	1	0	0

Comments on Interaction Matrices

- ▶ If you can't decide whether requirements conflict, you should assume that a conflict exists.
- ▶ If an error is made it is usually fairly cheap to fix; it can be much more expensive to resolve undetected conflicts.
- ▶ In the example, we are considering, we can see that R1 overlaps with R3 and conflicts with R5 and R6.
 - ▶ R2 is an independent requirement.
 - ▶ R3 overlaps with R1, R4, and R6.
- ▶ The advantage of using numeric values for conflicts and overlaps is that you can sum each row and column to find the number of conflicts and the number of overlaps.
- ▶ requirements which have high values for one or both of these figures should be carefully examined.

-
- ▶ A large number of conflicts or overlaps means that any
 - ▶ changes to that requirement will probably have a major impact on the rest of the requirements
 - ▶ Interaction matrices work only when there is relatively small number of requirements, as each requirement is compared with every other requirement
 - ▶ The upper limit should be about 200 requirements.
 - ▶ These overlaps and conflicts have to be discussed and resolved during requirements negotiation, which we'll discuss next.

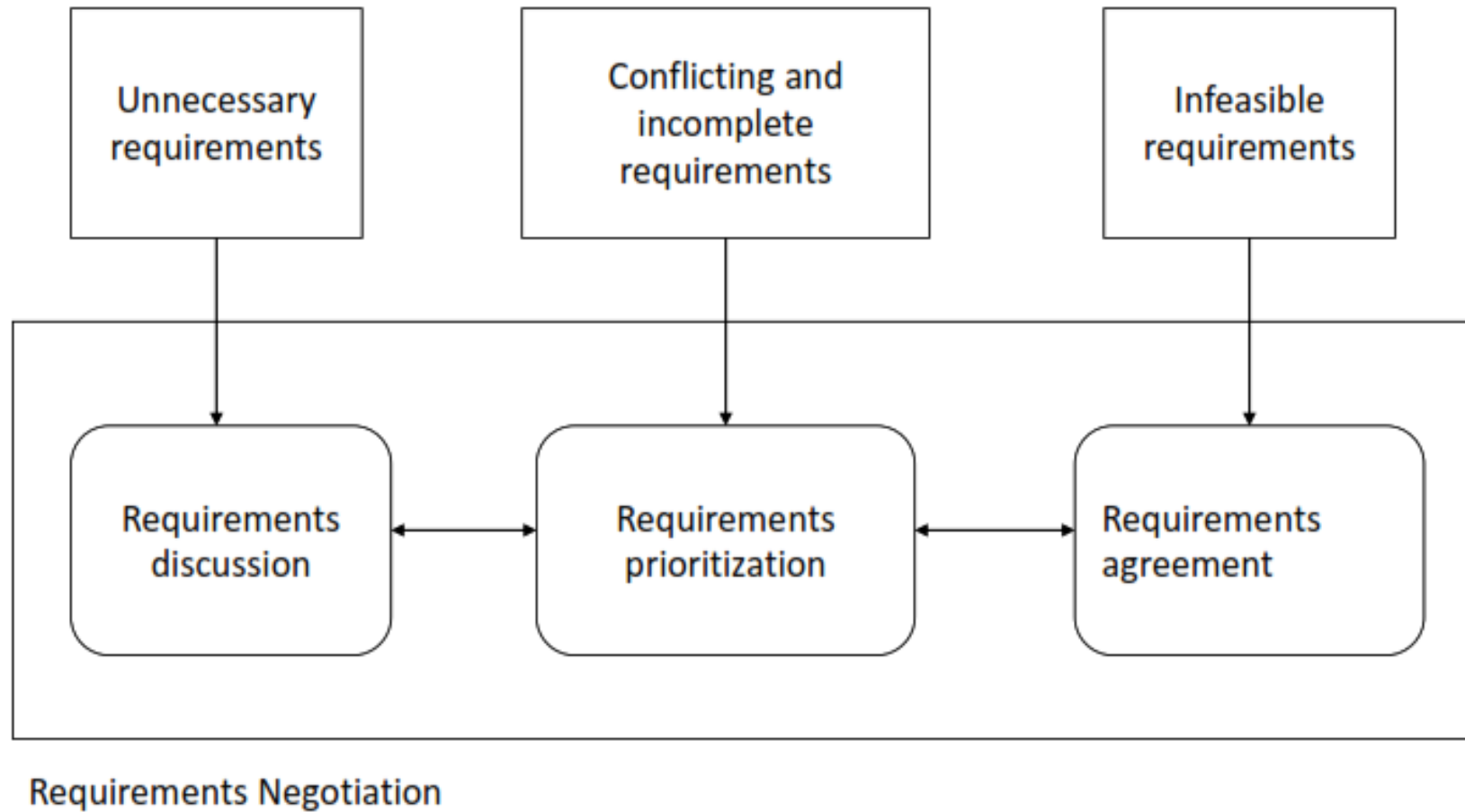
Requirements Negotiation

- ▶ Disagreements about requirements are inevitable when a system has
- ▶ many stakeholders.
- ▶ Conflicts are not 'failures' but reflect different stakeholder needs and
- ▶ priorities.
- ▶ Requirements negotiation is the process of discussing requirements conflicts and reaching a compromise that all stakeholders can agree to.
- ▶ In planning a requirements engineering process, it is important to leave
- ▶ enough time for negotiation.
- ▶ Finding an acceptable compromise can be time-consuming.
- ▶ The final requirements will always be a compromise which is governed by the needs of the organization in general, the specific requirements of different stakeholders, design and implementation constraints, and the budget and schedule for the system development.

▶ Requirements Negotiation Stages

- ▶ 1. Requirements Discussion
- ▶ 2. Requirements Prioritization
- ▶ 3. Requirements Agreement

Requirements Negotiation Process



Question

