3. REQUIREMENT ENGINEERING

Contents:

Requirement Engineering

- The process **of gathering the software requirements** from the client, analyze and then document.
- The process of **establishing the services that a customer requires from a system** and **the constraints** under which it operates and is developed.
- **The goal of requirement engineering:** to develop and maintain sophisticated and descriptive Software Requirement Specification (SRS) document.

What are software requirements?

- The system requirements **are the descriptions of the system services** and **constraints** under which it must operate.
- The software requirements are **description of features** and **functionalities** of the target system.
- Requirements convey the **expectations of users from the software** product.
- The requirements can be obvious or hidden, known or unknown, expected or unexpected from client’s point of view.
- A requirement in software engineering is a feature of new software that someone either wants, needs or commands.
- It describes **what the software does** and **any limitations** it should have.

Requirements Documents

- Requirements may serve a dual function:
    - As the basis of a bid for a contract
    - As the basis for the contract itself.
- “If a company wishes to let a contract for a large software development project it must define its needs in a sufficiently abstract way that a solution is not predefined.
- The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs.
- Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do.
- Both of these documents may be called the requirements document for the system

3.1 Functional and non-functional requirements

Functional requirements

- A functional requirement is a requirement that describes what the software does. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.
- All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.
- For example, in a hospital management system, a doctor should be able to retrieve the information of his patients.
- We express functional requirements in terms of:
    - o Data storage.
    - o Any processes that transform data.
    - o Any outputs that it can produce.

Functional requirements examples

- Functional requirements need to be clear, simple, and unambiguous. Here are some examples of well-written functional requirements:
  - The system must send a confirmation email whenever an order is placed.
  - The system must allow blog visitors to sign up for the newsletter by leaving their email.
  - The system must allow users to verify their accounts using their phone number.

User Story Vs Functional requirements examples

- Contrary to a popular misconception, functional requirements are not analogous to user stories, but stories can be a useful tool for deriving requirements with the user in mind. For example:
- **User story**: As an existing user, I want to be able to log into my account.
- **Functional requirements**:
  - The system must allow users to log into their account by entering their email and password.
  - The system must allow users to log in with their Google accounts.
  - The system must allow users to reset their password by clicking on "I forgot my password" and receiving a link to their verified email address.

Non-functional requirements

- A non-functional requirement defines any limitations that the software may have.
- These are basically the quality constraints that the system must satisfy according to the project contract.
- Nonfunctional requirements, not related to the system functionality, rather define how the system should perform
- The priority or extent to which these factors are implemented varies from one project to other.
- They are also called non-behavioral requirements.
- They basically deal with issues like: Portability, Security, Maintainability, Reliability, Scalability, Performance, Reusability, Flexibility
- For example, if a user prefers their software to have a larger amount of data storage, they may choose a software system that has a nonfunctional requirement that involves more storage space.

Non-functional requirements

- We express non-functional requirements in terms of the following:
  - Performance: for example, the number of transactions the software should do daily.
  - Security and access: it should comply with the law.
  - Technical constraint: run on an existing network.
  - Project constraint: the software should be ready within a set period.
  - Organizational constraint: the software should be teachable to new staff in a short amount of time.
  - Usability and reliability issues.

Non-functional requirements

- NFR’s are classified into following types:
  - Interface constraints
  - Performance constraints: response time, security, storage space, etc.
  - Operating constraints
  - Life cycle constraints: maintainability, portability, etc.
  - Economic constraints

Non-functional requirements

Performance and scalability. How fast does the system return results? How much will this performance change with higher workloads?

Portability and compatibility. Which hardware, operating systems, and browsers, along with their versions does the software run on? Does it conflict with other applications and processes within these environments?

Reliability, maintainability, availability. How often does the system experience critical failures? How much time does it take to fix the issue when it arises? And how is user availability time compared to downtime?

Security. How well are the system and its data protected against attacks?

Localization. Is the system compatible with local specifics?

Usability. How easy is it for a customer to use the system?

Non-functional requirements

Example of performance requirements:

The landing page supporting 5,000 users per hour must provide 6 second or less response time in a Chrome desktop browser, including the rendering of text and images and over an LTE connection.

Example of scalability requirements:

The system must be scalable enough to support 1,000,000 visits at the same time while maintaining optimal performance.

Non-functional requirements

Example of portability requirements:

A program running on Windows 10 must be able to run on Windows 11 without any change in its behavior and performance.

- ***Example of compatibility requirements:***

  The iOS application must support iPhone devices running on OS versions:3.6
- *3.3*
- *3.4*
- *4.3*
- *2.3*

Non-functional requirements

Example of reliability requirements:

The system must perform without failure in 95 percent of use cases during a month.

Example of maintainability requirements:

The mean time to restore the system (MTTRS) following a system failure must not be greater than 10 minutes. MTTRS includes all corrective maintenance time and delay time.

Example of availability requirements:

The web dashboard must be available to US users 99.98 percent of the time every month during business hours EST.

Non-functional requirements

Example of security requirement:

The payment processing gateway must be PCI DSS compliant.

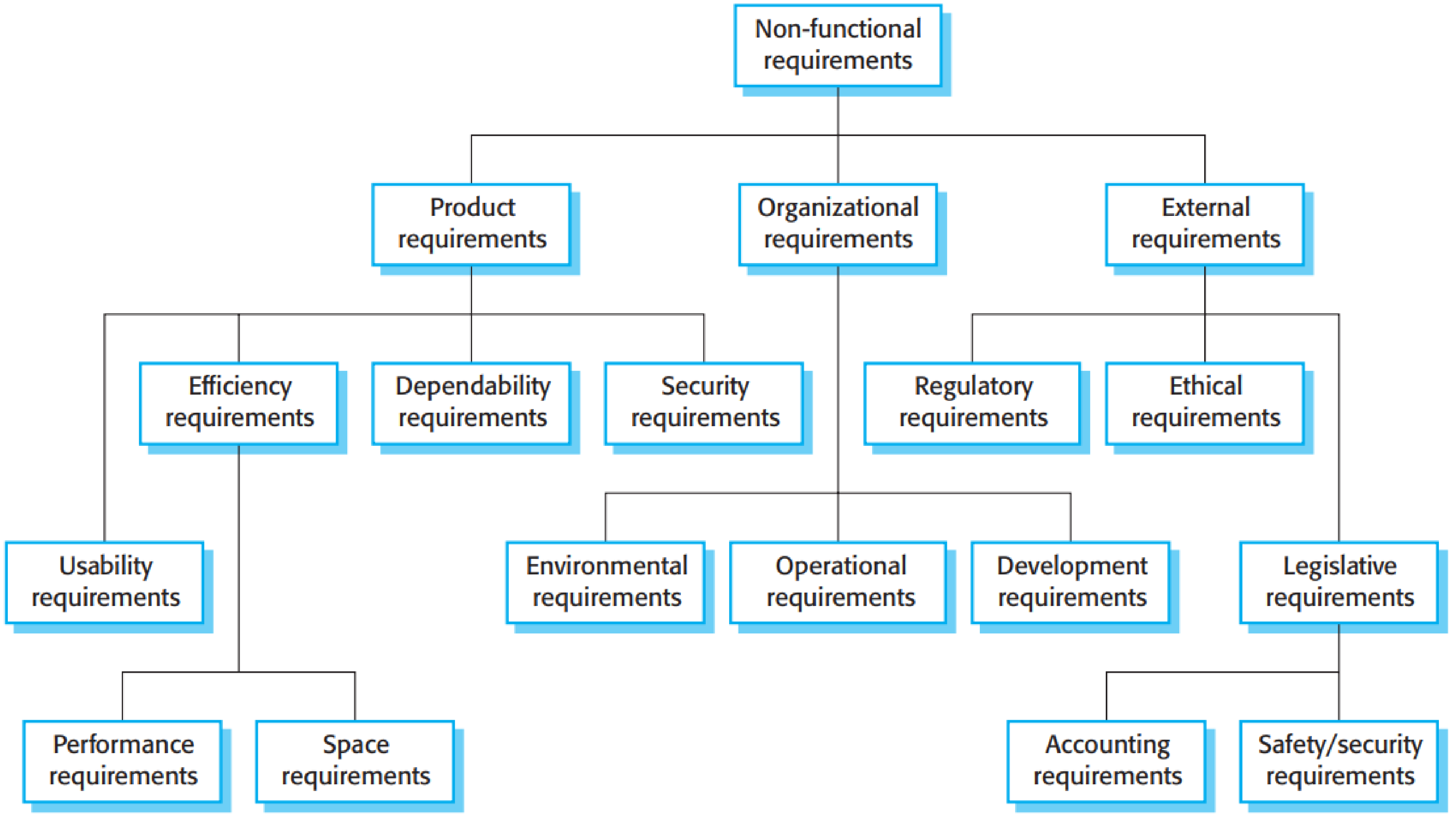
Example of a localization requirement:

The date format must be as follows: month.date.year.

Example of usability requirements:

The error rate of users submitting their payment details at the checkout page mustn't exceed 10 percent.

Non-functional requirements



Non-functional requirements

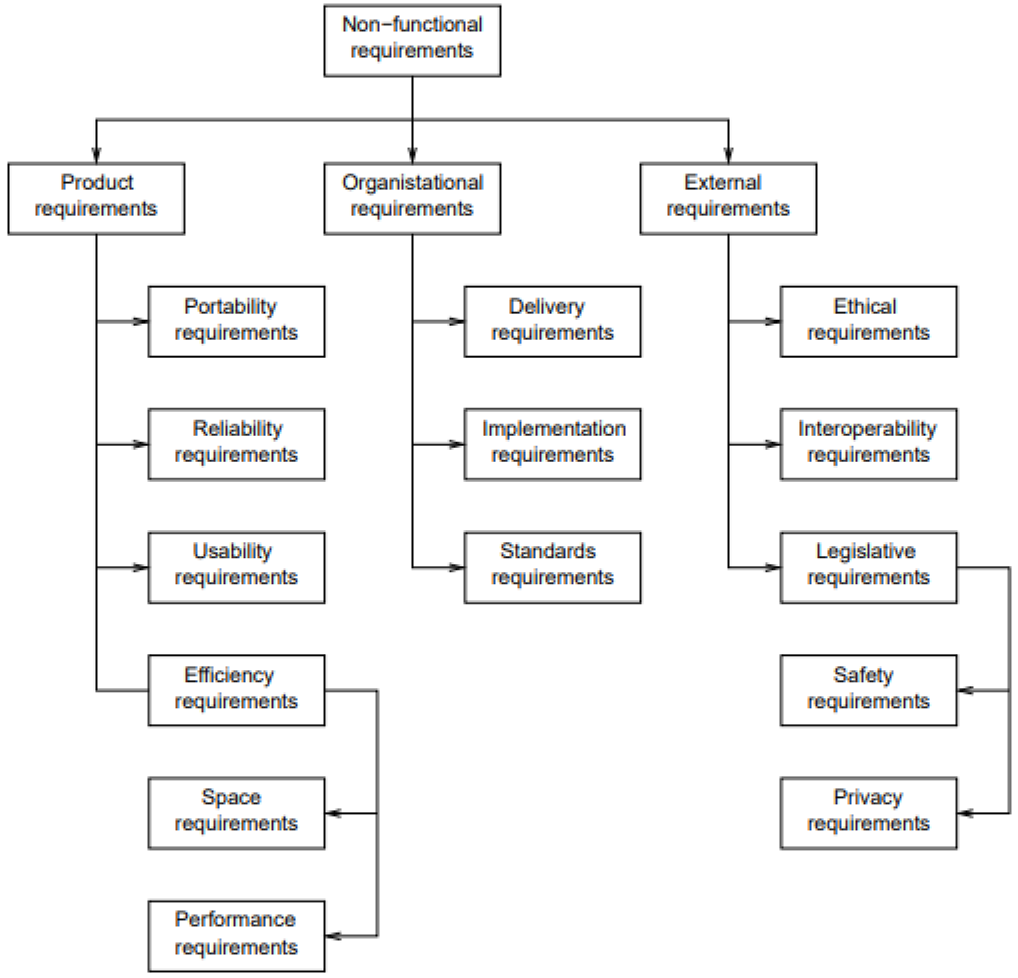


Figure 2.1: Non-functional Requirements

Non-functional requirements

- **Product requirements** – Requirements which specify that the delivered product must behave in a particular way, e.g. execution speed, reliability etc.
- **Organizational requirements** – Requirements which are a consequence of organizational policies and procedures, e.g. process standards used, implementation requirements etc.
- **External requirements** – Requirements which arise from factors which are external to the system and its development process, e.g. interoperability requirements, legislative requirement

3.2 Software Requirement Specification (SRS) documents

3.2 Software Requirement Specification (SRS) documents

- The software requirements document is the official statement of what is required of the system developers.
- Should **include both a definition** of user requirements and a **specification** of the system requirements.
- It is NOT a design document.
- As far as possible, it should state WHAT the system should do rather than HOW it should do it

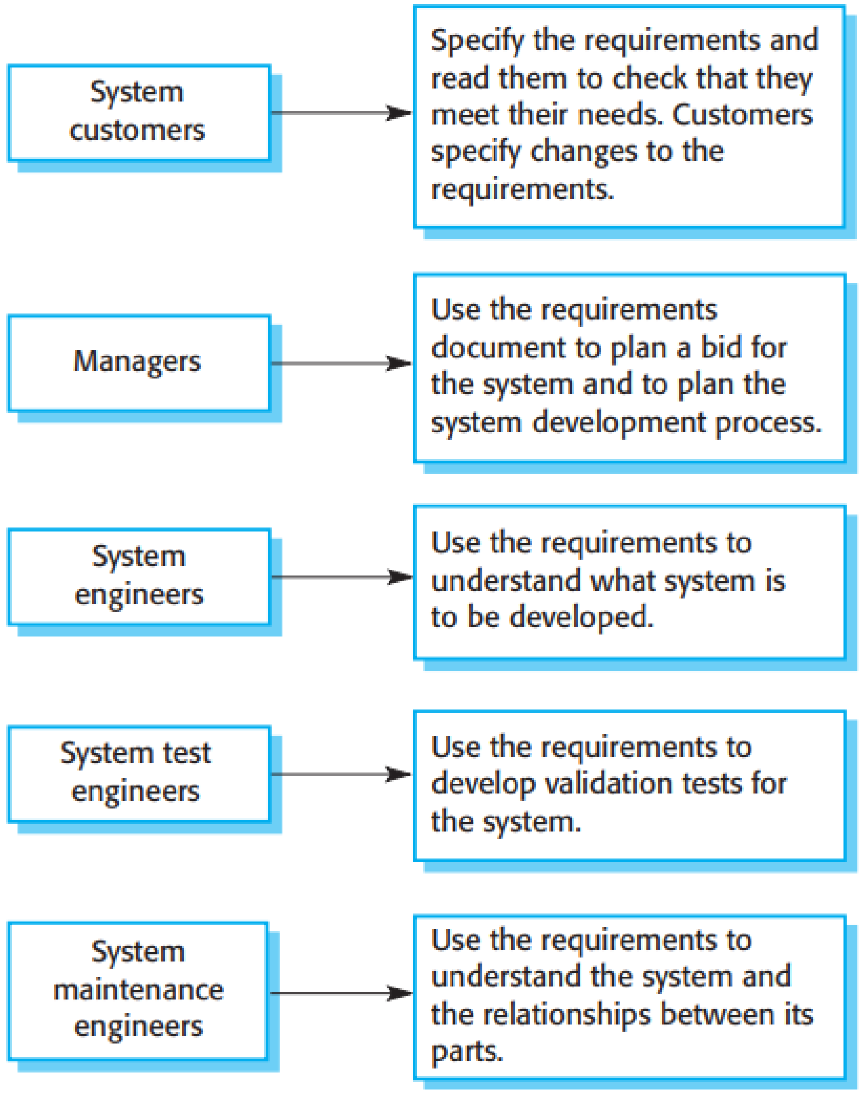
3.2 Software Requirement Specification (SRS) documents

Should: –

- specify external system behaviour
- specify implementation constraints
- be easy to change (but changes must be managed)
- serve as a reference tool for maintenance
- record forethought about the life cycle of the system (i.e. predict changes)
- characterise responses to unexpected event

Who are the users of SRS?

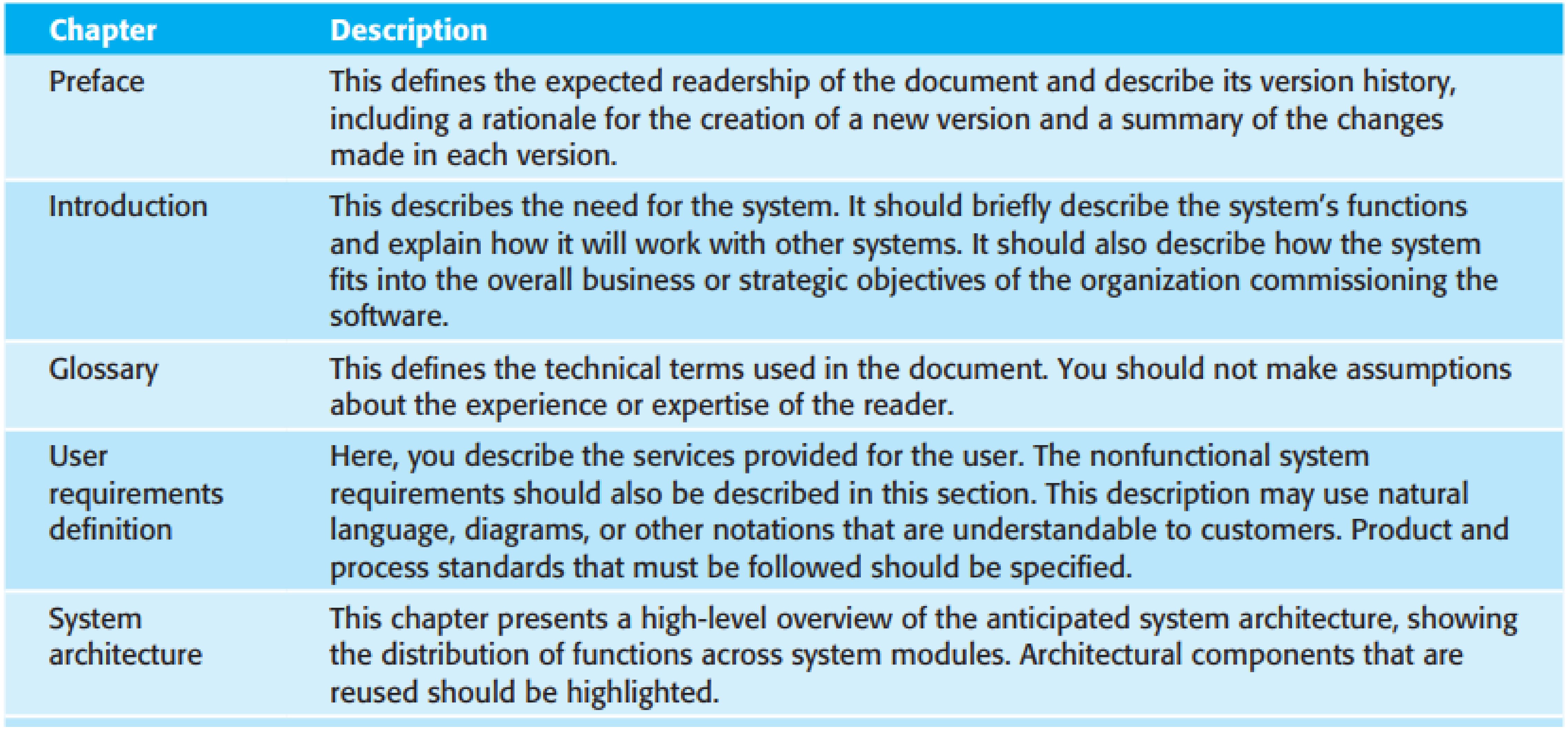
1. **System customers**
   - Specify the requirements and read them back to check that they meet their needs;
   - specify changes to the requirements
2. **Development Managers**
   - Use the requirements document to plan a bid for the system and to plan the system development process
3. **Implementation Programmers**
   - Use the requirements to understand what system is to be developed
4. **Test programmers**
   - Use the requirements to develop validation tests for the system
5. **Maintenance programmers**
   - Use the requirements to help understand the system and the relationships between its parts


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.
System test engineers
Use the requirements to develop validation tests for the system.
System maintenance engineers
Use the requirements to understand the system and the relationships between its parts.

The structure of a requirements document

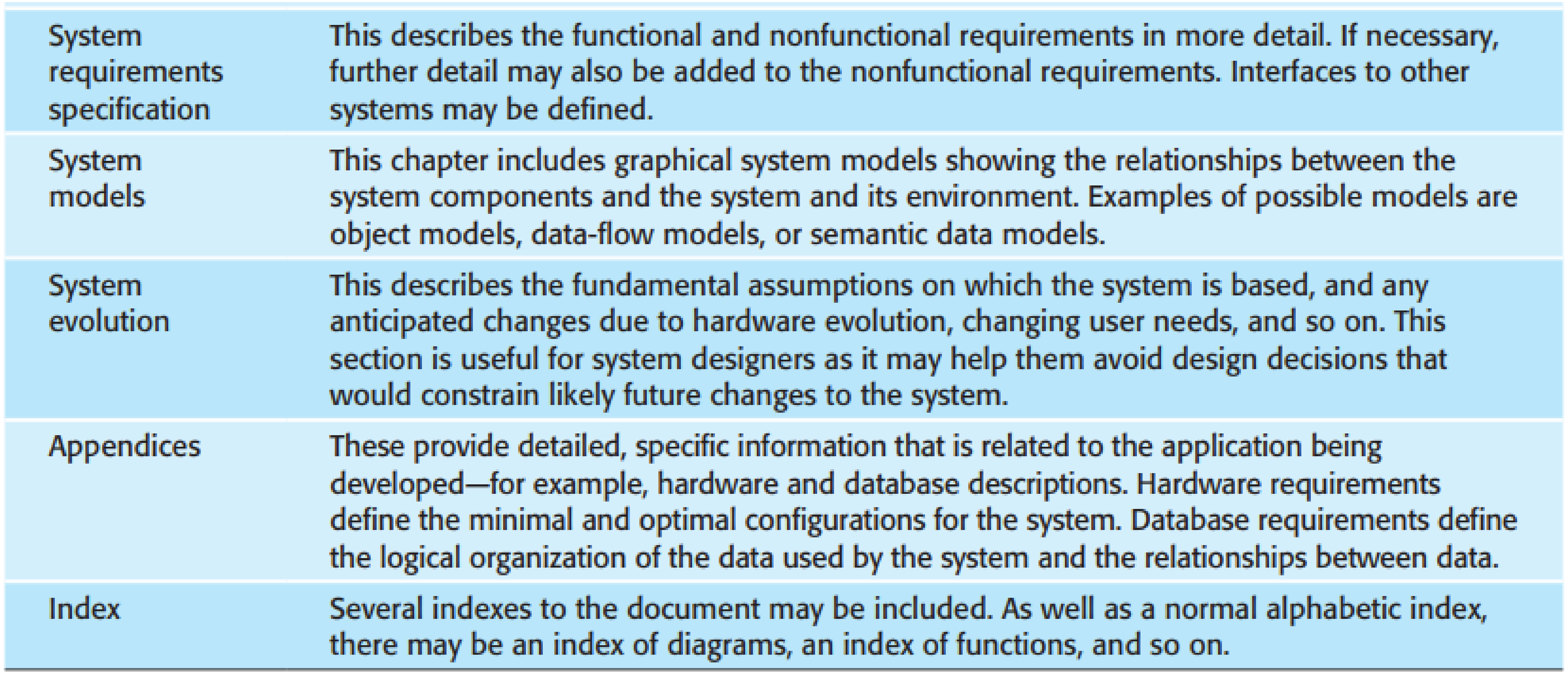
1. Introduction
2. Glossary
3. User requirements definition
4. System architecture
5. System requirements specification
6. System models
7. System evolution
8. Appendices
9. Index

The structure of a requirements document



Chapter	Description
Preface	This defines the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This describes the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This defines the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter presents a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

The structure of a requirements document



System requirements specification	This describes the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This chapter includes graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This describes the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These provide detailed, specific information that is related to the application being developed—for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Summary:

- Requirements set out what the system should do and define constraints on its operation and implementaion
- Functional requirements set out services that the system should provide
- Non-functional requirements constrain the system being developed or the development process
- User requirements are high-level statements of what the system should do
- User requirements should be written using natural language, tables and diagrams
- System requirements are intended to communicate the functions that the system should provide
- System requirements may be written in structured natural language, a PDL or in a formal language
- A software requirements document is an agreed statement of the system requirements.

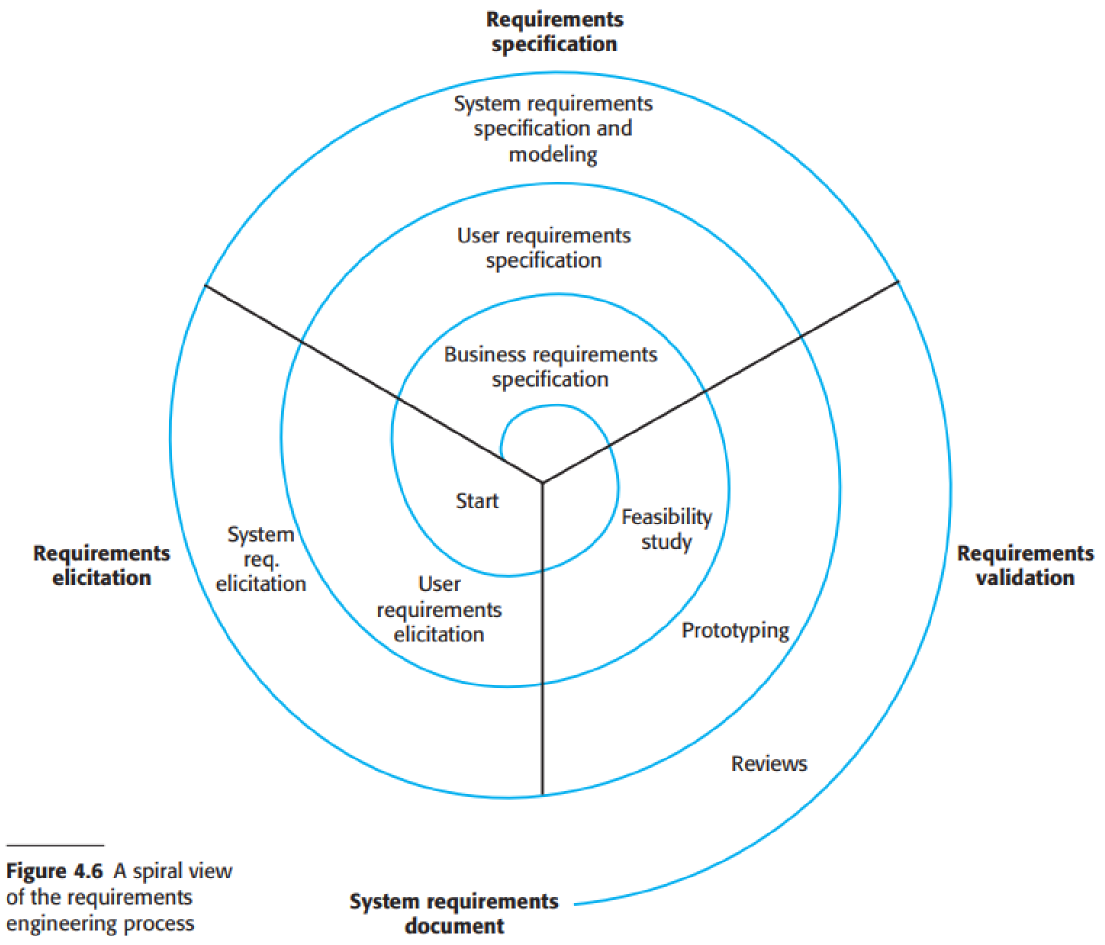
3.3 Requirement engineering process

3.3 Requirement engineering process

- The processes used for RE vary widely depending on the application domain, the people involved and the organization developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- In practice, RE is an iterative activity in which these processes are interleaved

A spiral view of the requirements engineering process

- Requirements engineering involves three key activities.
  - These are discovering requirements by interacting with stakeholders (**elicitation and analysis**);
  - converting these requirements into a standard form (**specification**); and
  - checking that the requirements actually define the system that the customer wants (**validation**).
- However, in practice, requirements engineering is an iterative process in which the activities are interleaved.
- Figure aside shows this interleaving.
- The activities are organized as an iterative process around a spiral. The output of the RE process is a system requirements document.



Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
Start
Feasibility study
Requirements elicitation
System req. elicitation
User requirements elicitation
Prototyping
Requirements validation
Reviews
System requirements document

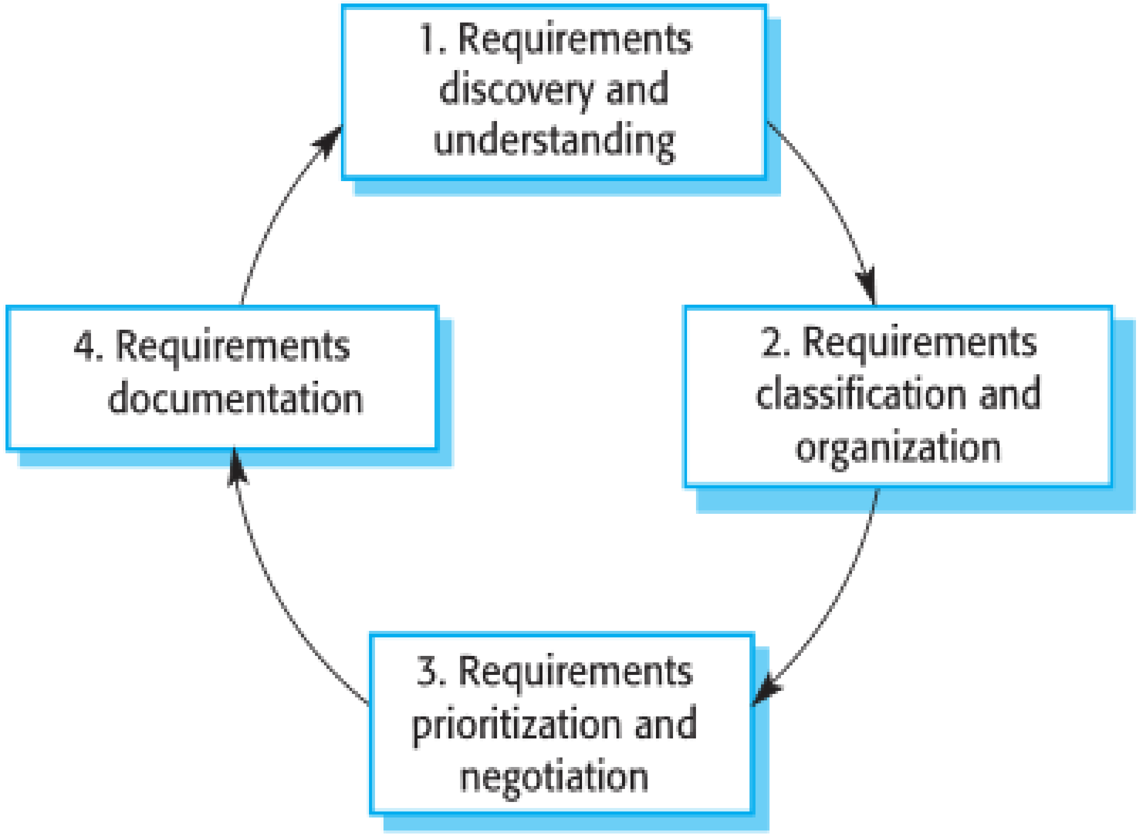
Figure 4.6 A spiral view of the requirements engineering process

Requirement Elicitation

- The aims of the requirements elicitation process are to understand the work that stakeholders do and how they might use a new system to help support that work.
- During requirements elicitation, software engineers work with stakeholders to find out about the application domain, work activities, the services and system features that stakeholders want, the required performance of the system, hardware constraints, and so on.
- The process activities are:
  1. Requirements discovery and understanding
  2. Requirements classification and organization
  3. Requirements prioritization and negotiation
  4. Requirements documentation

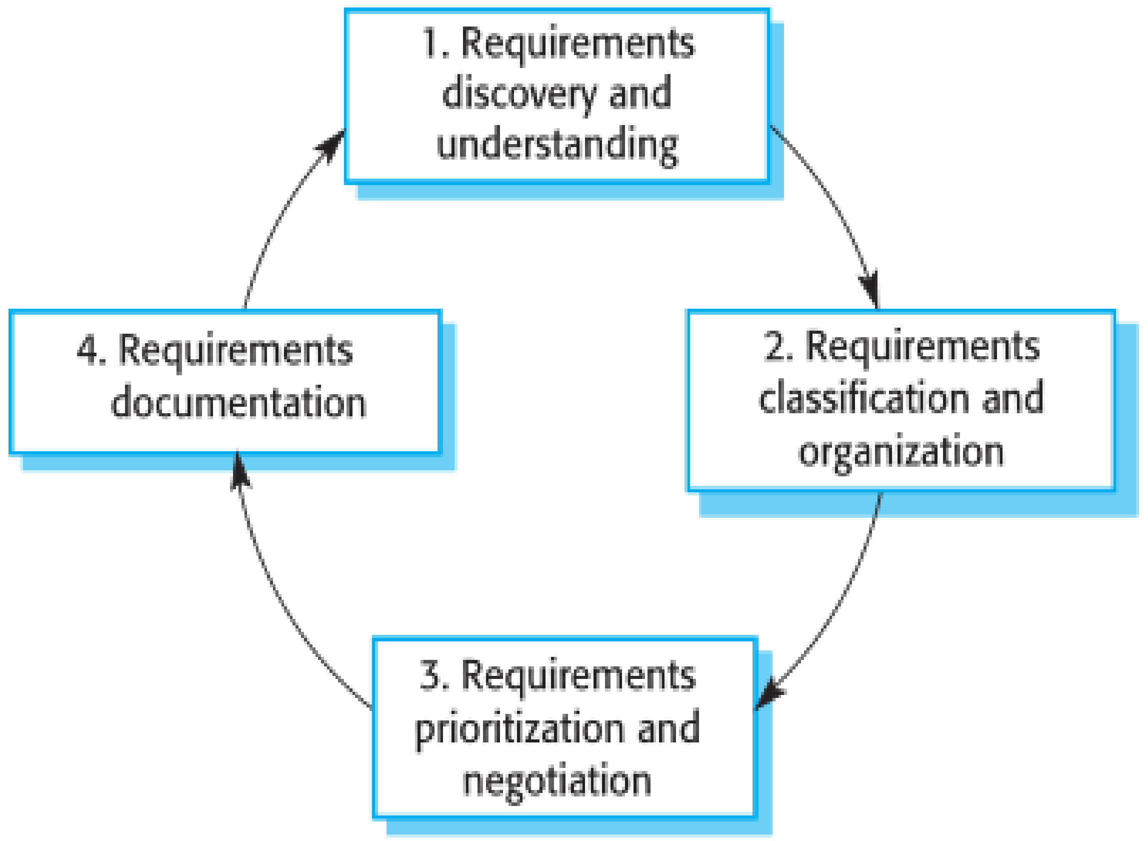
3.3 Requirement engineering process

1. **Requirements discovery and understanding**
   - This is the process of interacting with stakeholders of the system to discover their requirements.
   - Domain requirements from stakeholders and documentation are also discovered during this activity.
2. **Requirements classification and organization**
   - This activity takes the unstructured collection of requirements, groups related requirements and organizes them into coherent clusters.


1. Requirements discovery and understanding
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements documentation

3.3 Requirement engineering process

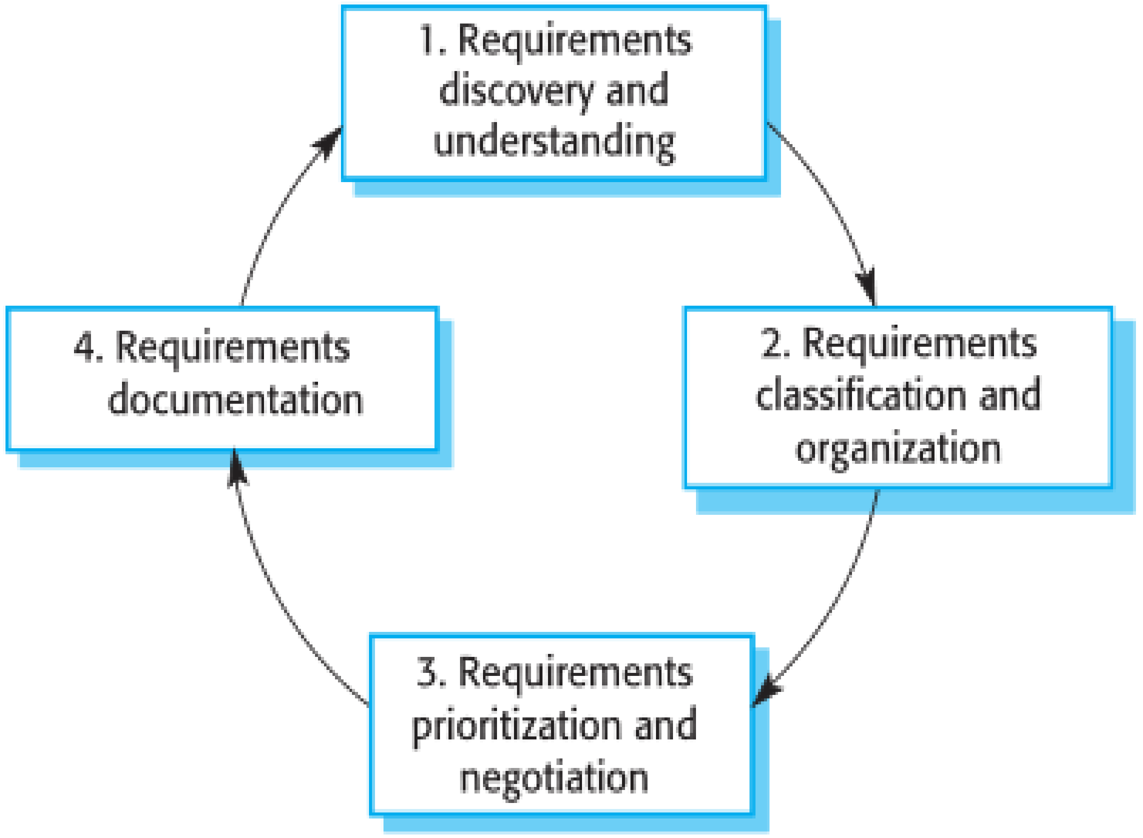
3. **Requirements prioritization and negotiation**
   - Inevitably, when multiple stakeholders are involved, requirements will conflict.
   - This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation.
   - Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.


1. Requirements discovery and understanding
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements documentation

3.3 Requirement engineering process

4. Requirements documentation

- The requirements are documented and input into the next round of the spiral.
- An early draft of the software requirements documents may be produced at this stage, or the requirements may simply be maintained informally on whiteboards, wikis, or other shared spaces.


1. Requirements discovery and understanding
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements documentation

Requirement Elicitation techniques

- Requirements elicitation involves meeting with stakeholders of different kinds to discover information about the proposed system.
- You need to spend time understanding how people work, what they produce, how they use other systems, and how they may need to change to accommodate a new system.
- There are two fundamental approaches to requirements elicitation:
    1. **Interviewing**, where you talk to people about what they do.
    2. **Observation or ethnography**, where you watch people doing their job to see what artifacts they use, how they use them, and so on.
- Besides these, there are other techniques such as survey, brainstorming, field visits, and so on.

Interviewing

- Formal or informal interviews with system stakeholders are part of most requirements engineering processes.
- In these interviews, the requirements engineering team puts questions to stakeholders about the system that they currently use and the system to be developed.
- Requirements are derived from the answers to these questions.
- Interviews may be of two types:
  - Closed interviews, where the stakeholder answers a predefined set of questions.
  - Open interviews, in which there is no predefined agenda. The requirements
- engineering team explores a range of issues with system stakeholders and hence develops a better understanding of their needs.

Problems with interviews:

- Application specialists may use language to describe their work that isn’t easy for the requirements engineer to understand.
- Interviews are not good for understanding domain requirements
  - Requirements engineers cannot understand specific domain terminology;
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn’t worth articulating

Ethnography

- Ethnography is an observational technique that can be used to understand operational processes and help derive requirements for software to support these processes.
- An analyst immerses himself or herself in the working environment where the system will be used.
- The day-to-day work is observed, and notes are made of the actual tasks in which participants are involved.
- The value of ethnography is that it helps discover implicit system requirements that reflect the actual ways that people work, rather than the formal processes defined by the organization.
- Ethnography is particularly effective for discovering two types of requirements:
    1. Requirements derived from the way in which people actually work, rather than the way in which business process definitions say they ought to work.
    2. Requirements derived from cooperation and awareness of other people’s activities.

Problems of requirements elicitation

- Stakeholders don’t know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organizational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

3.4 Requirements management

3.4 Requirements management

3.5 Business/Domain Modeling

3.5 Business/Domain Modeling

3.6 Business Process Reengineering(BPR)

3.6 Business Process Reengineering(BPR)

To be continued……