Requirements Engineering

- Functional and non-functional requirements
- Requirements specification
- The software requirements specification document (SRS)
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

Requirements Engineering

- The process of establishing the *services* that the customer requires from a system and the *constraints* under which it operates and is developed.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

Requirements Engineering

What is a requirement?

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

- Requirements engineering is the process of eliciting, documenting, analyzing, validating, and managing requirements.

Types of requirement

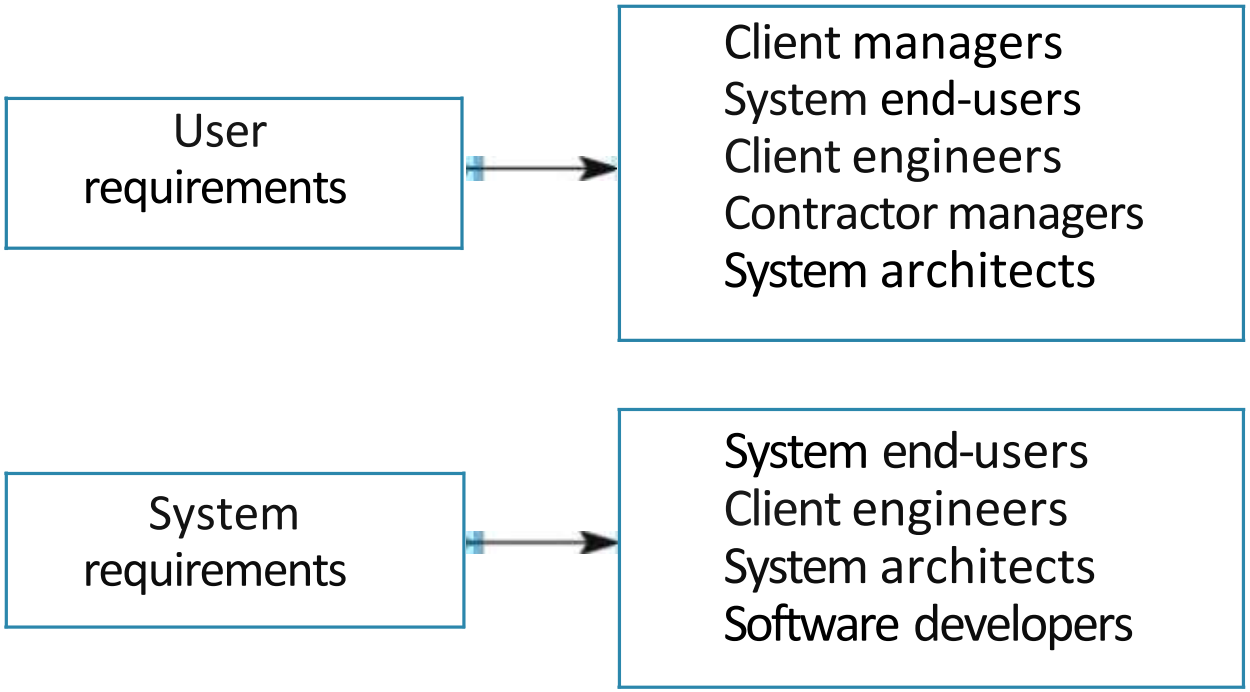
User requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

System requirements

- A structured document setting out detailed descriptions of the system's functions, services and operational constraints.
- Defines what should be implemented so may be part of a contract between client and contractor.

Readers of different types of requirements specification



Requirements Specification

Within these three specification types are there different requirements types?

- Functional requirements
- Nonfunctional requirements
- Domain requirement

Requirements Specification

What are functional requirements?

Functional requirements describe the services the system should provide. Functional requirements can be high-level and general or detailed, expressing inputs, outputs, exceptions, and so on.

What are nonfunctional requirements?
Nonfunctional requirements are imposed by the environment in which the system is to exist. These requirements could include timing constraints, quality properties, standard adherence, programming languages to be used, compliance with laws, and so on.

What are domain requirements?
Domain requirements are a type of nonfunctional requirement from which the application domain dictates or derives. Domain requirements might impose new functional requirements or constraints on existing functional requirements. For example: In the baggage inspection system, industry standards and restrictions on baggage size and shape will place certain constraints on the system.

Functional requirements

- Describe *functionality* or system *services.*
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Functional requirements

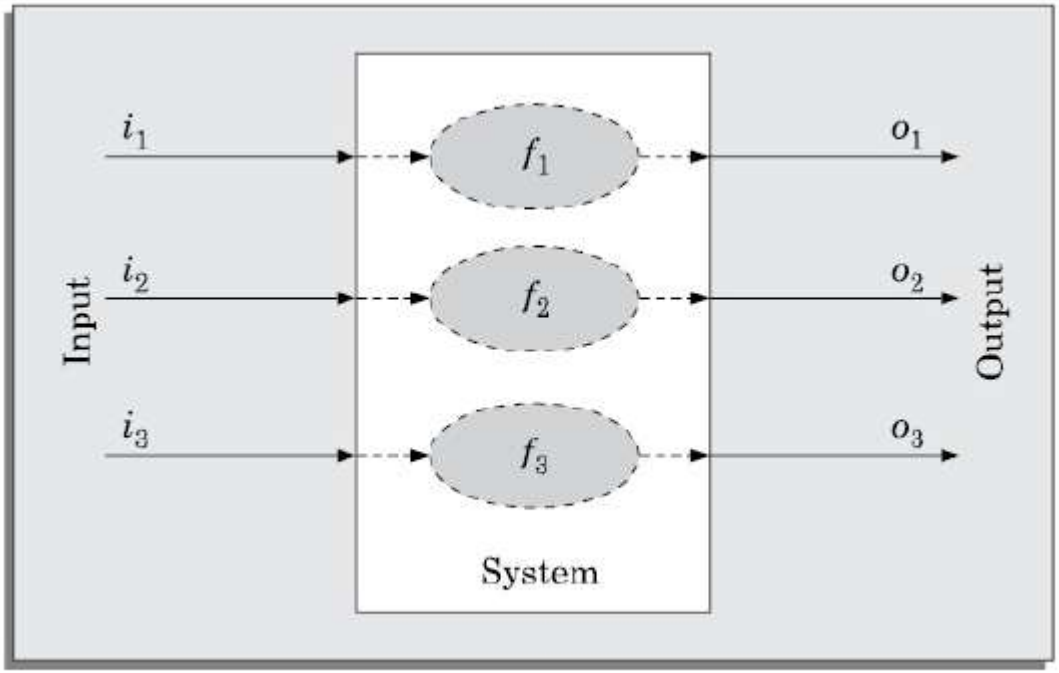


Figure 4.1: The black-box view of a system as performing a set of functions.

Non-functional requirements

- These define *system properties* and *constraints* e.g. reliability, response time, and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Metrics for specifying non-functional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes 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

Domain requirements

- The system's operational domain imposes requirements on the system.
  - For example, a train control system has to take into account the braking characteristics in different weather conditions.
- Domain requirements can be new functional requirements, constraints on existing requirements, or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Requirements Analysis

- The main purpose of the requirements analysis activity is to analyse the gathered requirements to remove all ambiguities, incompleteness, and inconsistencies from the gathered customer requirements and to obtain a clear understanding of the software to be developed.

During requirements analysis,the analyst needs to identify and resolve three main types of problems in the requirements:

- Anomaly
- Inconsistency
- Incompleteness

Agile methods and requirements

- Many agile methods argue that producing a requirements document is a waste of time as requirements change so quickly
- The document is therefore always out of date
- Methods such as XP use incremental requirements engineering and express requirements as ‘user stories’.

The software requirements document

The goal of the requirements analysis and specification phase is to clearly understand the customer requirements and to systematically organise the requirements into a document called the Software Requirements Specification (SRS) document.

Characteristics of a Good SRS Document

Concise: The SRS document should be concise and at the same time unambiguous, consistent, and complete.

Implementation-independent: The SRS should be free of design and implementation decisions unless those decisions reflect actual requirements. It should only specify what the system should do and refrain from stating how to do these. This means that the SRS document should specify the externally visible behavior of the system and not discuss the implementation issues.

Traceable: It should be possible to trace a specific requirement to the design elements that implement it and *vice versa*. Traceability is also important to verify the results of a phase with respect to the previous phase and to analyse the impact of changing a requirement on the design elements and the code.

Characteristics of a Good SRS Document

Modifiable: Customers frequently change the requirements during the software development development due to a variety of reasons. To cope up with the requirements changes, the SRS document should be easily modifiable. For this, an SRS document should be well-structured. A wellstructured
document is easy to understand and modify.

Identification of response to undesired events: The SRS document should discuss the system responses to various undesired events and exceptional conditions that may arise.

Verifiable: All requirements of the system as documented in the SRS document should be verifiable. This means that it should be possible to design test cases based on the description of the functionality as to whether or not requirements have been met in an implementation.

.

The software requirements document

An SRS document should clearly document the following aspects of a software:

- Functional requirements
- Non-functional requirements
  - — Design and implementation constraints
  - — External interfaces required
  - — Other non-functional requirements
- Goals of implementation.

The software requirements document

The **functional requirements** of the system, should clearly describe each functionality that the system would support along with the corresponding input and output data set.

Non-functional requirements usually address aspects concerning external interfaces, user interfaces, maintainability, portability, usability, maximum number of concurrent users, timing, and throughput (transactions per second, etc.).

The **goals of implementation** section might document issues such as easier revisions to the system functionalities that may be required in the future, easier support for new devices to be supported in the future, reusability issues, etc. These are the items which the developers might keep in their mind during development so that the developed system may meet some aspects that are not required immediately.

.

Functional requirements

This section can classify the functionalities either based on the specific functionalities invoked by different users.

1. User class 1
(a) Functional requirement 1.1
(b) Functional requirement 1.2

2. User class 2
(a) Functional requirement 2.1
(b) Functional requirement 2.2

Non-functional requirements

Performance requirements: Aspects such as number of transaction to be completed per second should be specified here. Some performance requirements may be specific to individual functional requirements or features. These should also be specified here.

Safety requirements: Those requirements that are concerned with possible loss or damage that could result from the use of the software are specified here. For example, recovery after power failure, handling software and hardware failures, etc. may be documented here.

Security requirements: This section should specify any requirements regarding security or privacy requirements on data used or created by the software. Any user identity authentication requirements should be described here. It should also refer to any external policies or regulations concerning the security issues. Define any security or privacy certifications that must be satisfied.

.

External interface requirements

User interfaces: This section should describe a high-level description of various interfaces and various principles to be followed. The user interface description may include sample screen images, any GUI standards or style guides that are to be followed, screen layout constraints, standard push buttons (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, etc. The details of the user interface design should be documented in a separate user interface specification document.

Hardware interfaces: This section should describe the interface between the software and the hardware components of the system. This section may include the description of the supported device types, the nature of the data and control interactions between the software and the hardware, and the communication protocols to be used.

External interface requirements

Software interfaces: This section should describe the connections between this software and other specific software components, including databases,. operating systems, tools, libraries, and integrated commercial components, etc. Identify the data items that would be input to the software and the data that would be output should be identified and the purpose of each should be described.

Communications interfaces: This section should describe the requirements associated with any type of communications required by the software, such as e-mail, web access, network server communications protocols, etc.

Overall description of organisation of SRS document

Introduction

Purpose: This section should describe where the software would be deployed and and how the software would be used.

Project scope: This section should briefly describe the overall context within which the software is being developed.

Environmental characteristics: This section should briefly outline the environment (hardware and other software) with which the software will interact.

Overall description of organisation of SRS document

Product perspective: This section needs to briefly state as to whether the software is intended to be a replacement for a certain existing systems, or it is a new software. A simple schematic diagram can be given to show the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.

Product features: This section should summarize the major ways in which the software would be used

User classes: Various user classes that are expected to use this software are identified and described here. The different classes of users are identified by the types of functionalities that they are expected to invoke, or their levels of expertise in using computers.

Overall description of organisation of SRS document

Operating environment: This section should discuss in some detail the hardware platform on which the software would run, the operating system, and other application software with which the developed software would interact.

Design and implementation constraints: In this section, the different. constraints on the design and implementation are discussed.

User documentation: This section should list out the types of user documentation, such as user manuals, on-line help, and trouble-shooting manuals that will be delivered to the customer along with the software.

USE CASE DIAGRAM

Representation of Use Cases

1. A use case model can be documented by drawing a use case diagram and writing an accompanying text elaborating the drawing.
2. In the use case diagram, each use case is represented by an ellipse with the name of the use case written inside the ellipse.

3. All the ellipses (i.e. use cases) of a system are enclosed within a rectangle which represents the system boundary. The name of the system being modeled (e.g., library information system) appears inside the rectangle.

4. The different users of the system are represented by using stick person icons. Each stick person icon is referred to as an actor. An actor is a role played by a user with respect to the system use. It is possible that the same user may play the role of multiple actors.
5. An actor can participate in one or more use cases.
6. The line connecting an actor and the use case is called the communication relationship. It indicates that an actor makes use of the functionality provided by the use case.

USE CASE DIAGRAM

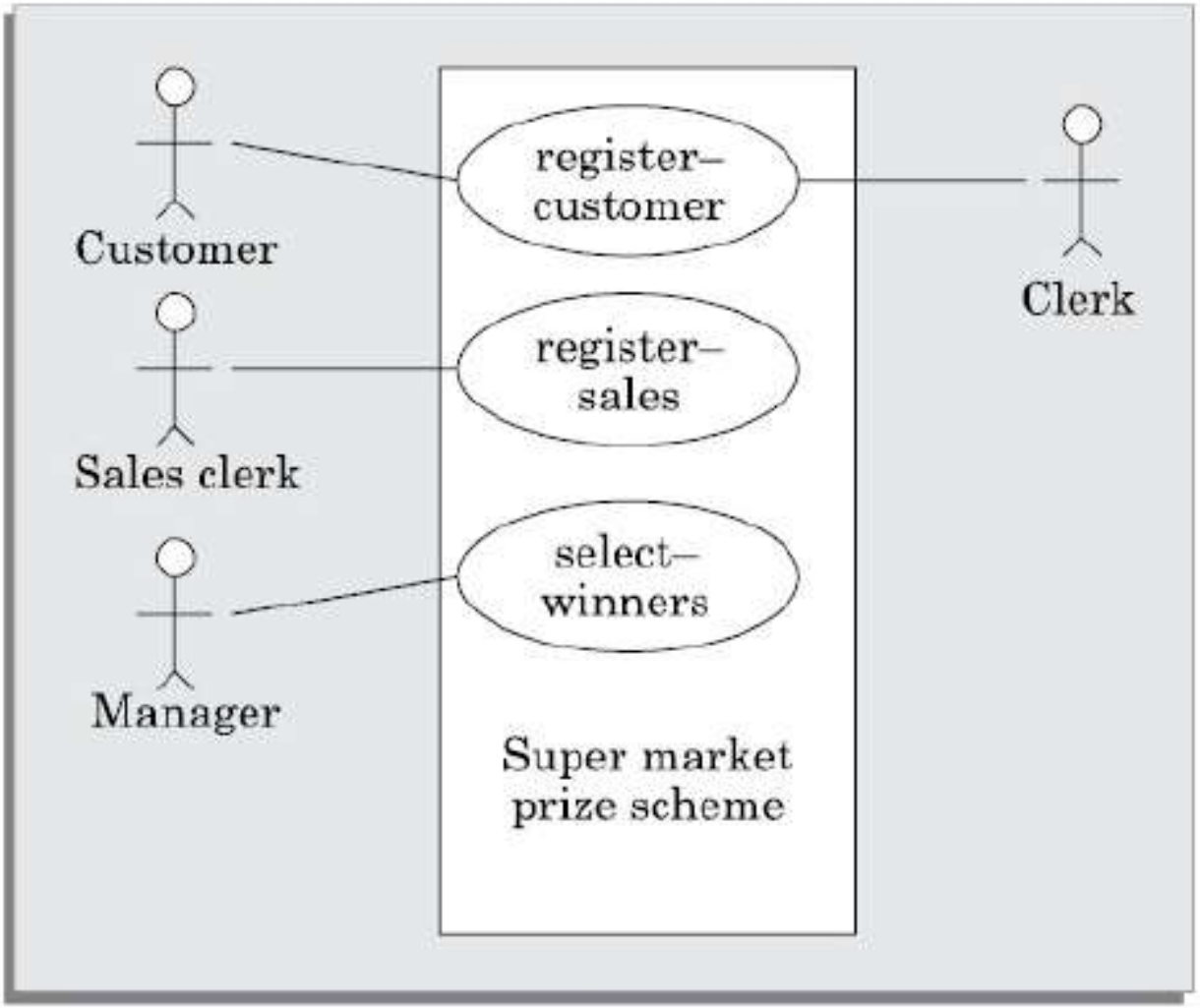
How to Identify the Use Cases of a System?

Identification of the use cases involves brain storming and reviewing the SRS document. Typically, the high-level requirements specified in the SRS document correspond to the use cases. In the absence of a wellformulated SRS document, a popular method of identifying the use cases is actor-based.

This involves first identifying the different types of actors and their usage of the system. Subsequently, for each actor the different functions that they might initiate or participate are identified.

For example, for a Library Automation System, the categories of users can be members, librarian, and the accountant. Each user typically focuses on a set of functionalities. Foe example, the member typically concerns himself with book issue, return, and renewal aspects. The librarian concerns himself with creation and deletion of the member and book records. The accountant concerns itself with the amount collected from membership fees and the expenses aspects.

USE CASE DIAGRAM



USE CASE DIAGRAM

Text description
U1: register-customer: Using this use case, the customer can register himself by providing the necessary details.
Scenario 1: Mainline sequence
1. Customer: select register customer option
2 . System: display prompt to enter name, address, and telephone number.
3. Customer: enter the necessary values
4: System: display the generated id and the message that the customer has successfully been registered.
Scenario 2: At step 4 of mainline sequence
4 : System: displays the message that the customer has already registered.
Scenario 3: At step 4 of mainline sequence
4 : System: displays message that some input information have not been entered. The system displays a prompt to enter the missing values.

USE CASE DIAGRAM

Text description

U2: register-sales: Using this use case, the clerk can register the details of the purchase made by a customer.
Scenario 1: Mainline sequence
1. Clerk: selects the register sales option.
2. System: displays prompt to enter the purchase details and the id of the customer.
3. Clerk: enters the required details.
4 : System: displays a message of having successfully registered the sale.
U3: select-winners. Using this use case, the manager can generate the winner list.
Scenario 2: Mainline sequence
1. Manager: selects the select-winner option.
2 . System: displays

Tutorial 1

Develop a Use Case Diagram for a real-life project under consideration.

Team Size 2
Submit a Hard Copy
Date and Time of Submission: 09/10/2024 , 9am

Requirements engineering processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation 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.

Requirements engineering processes

Requirements Elicitation

What is requirements elicitation?
Requirements elicitation involves working with customers to determine the application domain, the services that the system should provide, and the operational constraints of the system.
Elicitation may involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, and so on. These people are collectively called stakeholders.

What is a requirements validation?
Requirement validation is the process of checking and confirming that the requirements defined for development accurately capture the needs and expectations of the stakeholders. So now you know it's a systematic approach ensuring the requirements are clear, unambiguous, verifiable, and achievable.

Requirements engineering processes

Requirements Modeling

How are software requirements modeled?

There are a number of ways to model software requirements; these include natural languages, informal and semiformal techniques, user stories, use case diagrams, structured diagrams, object-oriented techniques, formal methods, and more.

What are user stories?

User stories are short conversational texts that are used for initial requirements discovery and project planning. User stories are widely used in conjunction with agile methodologies.

User stories are written by the customers in their own “voice,” in terms of what the system needs to do for them. User stories usually consist of two to four sentences written by the customer in his own terminology, usually on a three-by-five inch card. The appropriate amount of user stories for one system increment or evolution is about 80, but the appropriate number will vary widely depending upon the application size and scope.

A spiral view of the requirements engineering process