STUDY GUIDE FOR MODULE NO. 3

CHAPTER 3: SOFTWARE REQUIREMENTS ANALYSIS

MODULE OVERVIEW

In this module, we will explore requirements specification or also known as requirements engineering and to explain the processes involved by which requirements are elicited and defined. This is a particularly critical stage of the software process, as mistakes made at this stage inevitably lead to later problems in the system design and implementation.

This will also give you deeper knowledge and understanding on the different types of requirements and why these requirements should be written in different ways, as well as the significant differences of functional and non-functional requirements.

MODULE LEARNING OBJECTIVES

At the end of this chapter, the student should be able to:
1. Understand the concepts of user requirements and system requirements and why these requirements may be expressed using different notations;
2. Differentiate functional and non-functional requirements;
3. Understand the two techniques for describing systems requirements, namely structured natural language and programming language based descriptions;
4. Understand the main requirements engineering activities of elicitation, analysis, and validation, and the relationships between these activities.

LEARNING CONTENTS (TOPIC 1: REQUIREMENTS ENGINEERING)

WHAT IS SOFTWARE REQUIREMENTS SPECIFICATION?

As discussed from the previous module, this is the process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed. It is also known as ***Requirements Engineering***.

The **requirements** for a system are the descriptions of the services that a system should provide and the constraints on its operation. On the other hand, software specification / requirements engineering is the process of:

- Finding out,
- Analysing,
- Documenting,
- Checking these services and constraints.

The term ***requirement*** is not used consistently in the software industry. In some cases, a requirement is simply a high-level, abstract statement of a service that a system should provide or a constraint on a system. At the other extreme, it is a detailed, formal definition of a system function. Davis (Davis 1993) explains why these differences exist:

"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."

TYPES OF REQUIREMENTS

To make a clear separation between the different levels of description of the requirements, let us use the term ***user requirements*** to mean the high-level abstract statement and ***system requirements*** to mean the detailed description of what the system should do.

USER REQUIREMENTS VS. SYSTEM REQUIREMENTS

1. User Requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.

2. System Requirements are more detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

Different kinds of requirement are needed to communicate information about a system to different types of reader. Figure 3.1 illustrates the distinction between user and system requirements.

User requirement definition

1. The software must provide a means of representing and accessing external files created by other tools.

System requirements specification

1.1 The user should be provided with facilities to define the type of external files.
1.2 Each external file type may have an associated tool which may be applied to the file.
1.3 Each external file type may be represented as a specific icon on the user's display.
1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.
1.5 When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

Figure 3.1 Example User Requirements and System Requirements

As you can see from Figure above, the user requirement is quite general. The system requirements provide more specific information about the services and functions of the system that is to be implemented.

You need to write requirements at different levels of detail because different types of readers use them in different ways. Figure 3.2 shows the types of readers of the user and system requirements

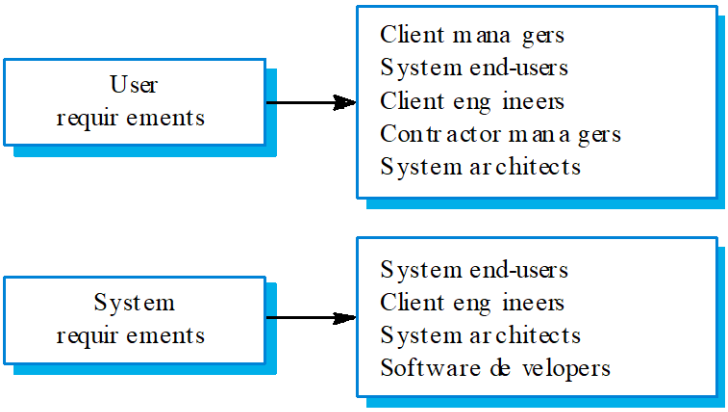

Figure 3.2 Readers of Requirements Specifications

The different types of document readers shown in Figure 3.2 are examples of system stakeholders. As well as users, many other people have some kind of interest in the system

FUNCTIONAL VS. NON-FUNCTIONAL REQUIREMENTS

Software system requirements are often classified as functional or non-functional requirements:

1. Functional requirements

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

Here are examples of Functional Requirements for a Mentcare System:

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

2. 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. These non-functional requirements usually specify or constrain characteristics of the system as a whole. They may relate to emergent system properties such as reliability, response time, and memory use. Alternatively, they may define constraints on the system implementation, such as the capabilities of I/O devices or the data representations used in interfaces with other systems.

NOTE: Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

Figure 3.3 is a classification of non-functional requirements. You can see from this diagram that the non-functional requirements may come from required characteristics of the software (product requirements), the organization developing the software (organizational requirements), or external sources:

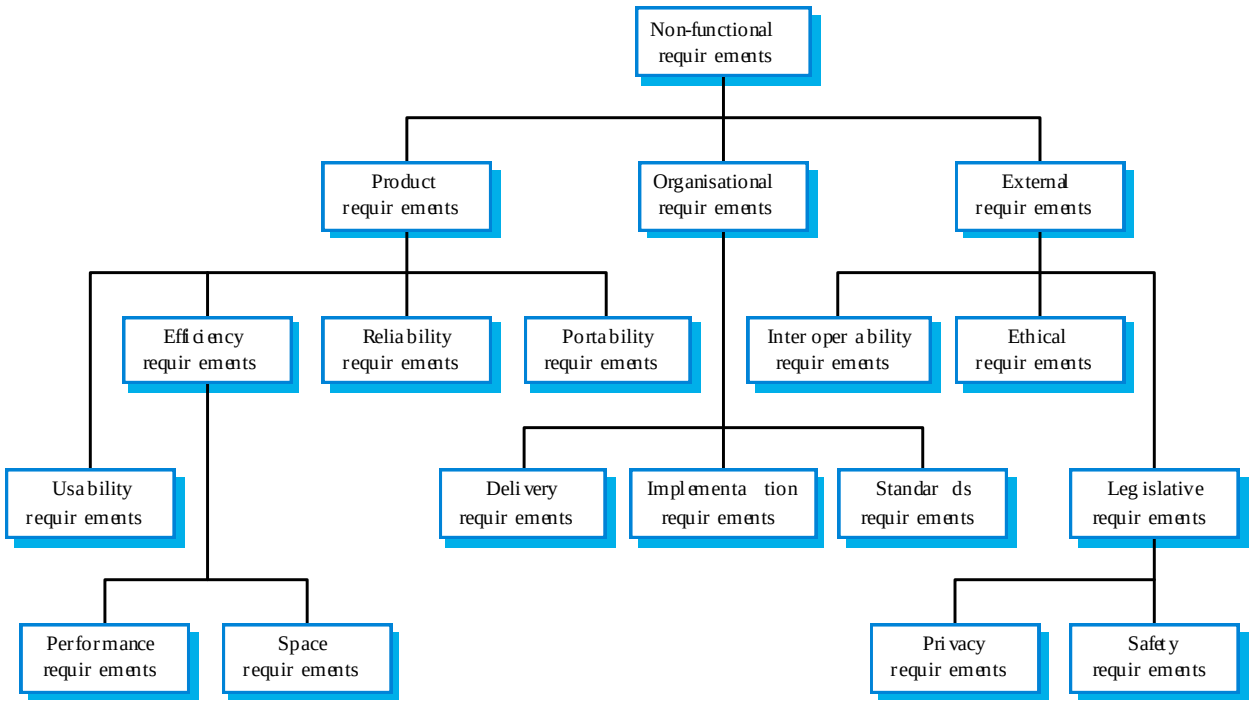

Figure 3.3 Types of Non-Functional Requirements

Non-Functional Classifications and examples for the Mentcare System:

A. Product requirements

- Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.

  Example: The Mentcare system shall be available to all clinics during normal working hours (Mon–Fri, 08:30–17:30). Downtime within normal working hours shall not exceed 5 seconds in any one day.

B. Organizational requirements

- Requirements which are a consequence of organizational policies and procedures e.g. process standards used, implementation requirements, etc.

  Example: Users of the Mentcare system shall identify themselves using their health authority identity card.

C. External requirements

- Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

  Example: The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Whenever possible, you should write non-functional requirements quantitatively so that they can be objectively tested. Table 3.1 shows metrics that you can use to specify non-functional system properties. You can measure these characteristics when the system is being tested to check whether or not the system has met its nonfunctional requirements.

PROPERTY	MEASURE
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Megabytes/Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failur
Portability	Percentage of target dependent statements Number of target systems

Table 3.1 Metrics for specifying nonfunctional requirements

It is difficult to separate functional and non-functional requirements in the requirements document. If the non-functional requirements are stated separately from the functional requirements, the relationships between them may be hard to understand. However, you should, ideally, highlight requirements that are clearly related to emergent system properties, such as performance or reliability. You can do this by putting them in a separate section of the requirements document or by distinguishing them, in some way, from other system requirements.

LEARNING CONTENTS (TOPIC 2: REQUIREMENTS ENGINEERING PROCESS)

WHAT ARE THE KEY ACTIVITIES OF SOFTWARE SPECIFICATIONS OR REQUIREMENTS ENGINEERING?

Requirements engineering is an important aspect of any software project, and is a general term used to encompass all the activities related to requirements. This involves three key activities:

1. **Requirements Elicitation & Analysis**
2. **Requirements Specification**
3. **Requirements Validation**

Although they seem to be separate tasks, these three processes cannot be strictly separated and performed sequentially. Some of the requirements are implicit in the working practices, while others may only arise when design solutions are proposed. Figure 3.4 shows the process and stages of requirements engineering.

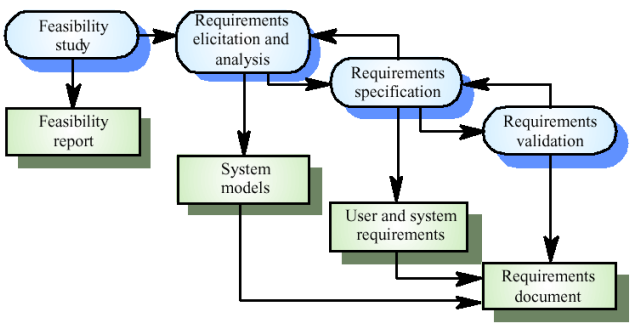

Figure 3.4 Requirements Engineering Process

A **feasibility study** decides whether or not the proposed system is worthwhile. It is a short focused study that checks

- If the system contributes to organisational objectives
- If the system can be engineered using current technology and within budget
- If the system can be integrated with other systems that are used

1. Requirements Elicitation & Analysis

- 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.

  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.

  You should use a mix of interviewing and observation to collect information and, from that, you derive the requirements, which are then the basis for further discussions.

2. Requirements Specification

- Requirements specification is the process of writing down the user and system requirements in a requirements document. Ideally, the user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent.

- *User requirements* are almost always written in natural language supplemented by appropriate diagrams and tables in the requirements document. *System requirements* may also be written in natural language, but other notations based on forms, graphical, or mathematical system models can also be used. Figure 3.5 summarizes possible notations for writing system requirements and other alternatives to Natural Language (NL) specifications.

NOTATION	DESCRIPTION
Natural language sentences	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system. UML (unified modeling language) use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want, and they are reluctant to accept it as a system contract.

Figure 3.5 Notations for Writing System Requirements

NOTE: If you are writing user requirements, you should not use software jargon, structured notations, or formal notations. You should write user requirements in natural language, with simple tables, forms, and intuitive diagrams

However, there are problems with Natural Language (NL) Specifications, namely:

- **Ambiguity** - The readers and writers of the requirement must interpret the same words in the same way. NL is naturally ambiguous so this is very difficult.
- **Over-flexibility** -The same thing may be said in a number of different ways in the specification.
- **Lack of Modularisation** -NL structures are inadequate to structure system requirements.

Here are some examples of Alternatives to Natural Language (NL) Specifications:

A. Form-based Node Specification (under Structured Natural Language)

Insulin Pump/Control Software/SRS/3.3.2	
Function	Compute insulin dose: Safe sugar level
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1)
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose Š the dose in insulin to be delivered
Destination	Main control loop
Action:	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requires	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin..
Post-condition	r0 is replaced by r1 then r1 is replaced by r2
Side-effects	None

Figure 3.6 The Structured Specification of a Requirement for an Insulin Pump

B. Tabular Specification (under Structured Natural Language)

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2-r1)<(r1-r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing. ((r2-r1) □ (r1-r0))	CompDose = round ((r2-r1)/4) If rounded result = 0 then CompDose = MinimumDose

Figure 3.7 The Tabular Specification of Computation in an Insulin Pump

C. Sequence Diagram (under Graphical Notations)

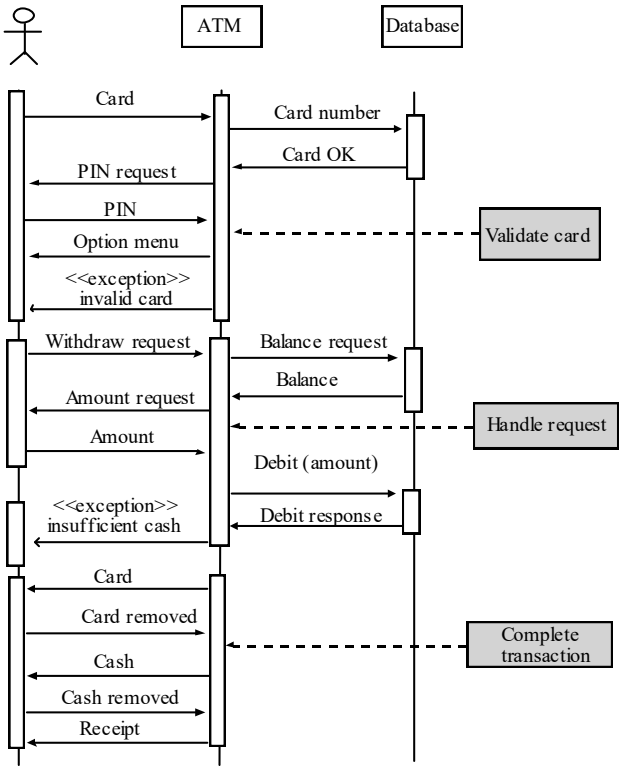

Figure 3.8 Sequence Diagram of ATM Withdrawal

D. Use-Case Diagram (under Graphical Notations)

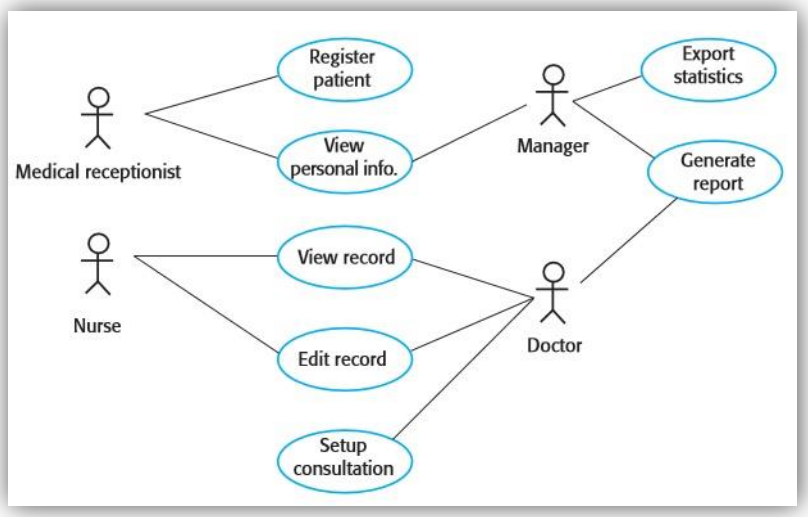

Figure 3.9 Use Case Diagram for the Mentcare System

3. Requirements Validation

- Requirements validation is the process of checking that requirements define the system that the customer really wants. It overlaps with elicitation and analysis, as it is concerned with finding problems with the requirements. Requirements validation is critically 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.

During the requirements validation process, different types of checks should be carried out on the requirements in the requirements document. These checks include:

1. Validity checks. These check that the requirements reflect the real needs of system users. Because of changing circumstances, the user requirements may have changed since they were originally elicited.

2. 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.

3. Completeness checks. The requirements document should include requirements that define all functions and the constraints intended by the system user.

4. Realism checks. By using knowledge of existing technologies, the requirements should be checked to ensure that they can be implemented within the proposed budget for the system. These checks should also take account of the budget and schedule for the system development.

5. Verifiability. To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable. This means that you should be able to write a set of tests that can demonstrate that the delivered system meets each specified requirement.

REQUIREMENTS VALIDATION TECHNIQUES

A number of requirements validation techniques can be used individually or in conjunction with one another:

1. Requirements Reviews - The requirements are analyzed systematically by a team of reviewers who check for errors and inconsistencies.

2. Prototyping - This involves developing an executable model of a system and using this with end-users and customers to see if it meets their needs and expectations. Stakeholders experiment with the system and feedback requirements changes to the development team.

3. 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.

LEARNING CONTENTS (TOPIC 3: THE SOFTWARE REQUIREMENTS DOCUMENT)

THE SOFTWARE REQUIREMENTS DOCUMENT

- The **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 set of WHAT the system should do rather than HOW it should do it.

The requirements document has a diverse set of users, ranging from the senior management of the organization that is paying for the system to the engineers responsible for developing the software. Figure 3.10 shows possible users of the document and how they use it.

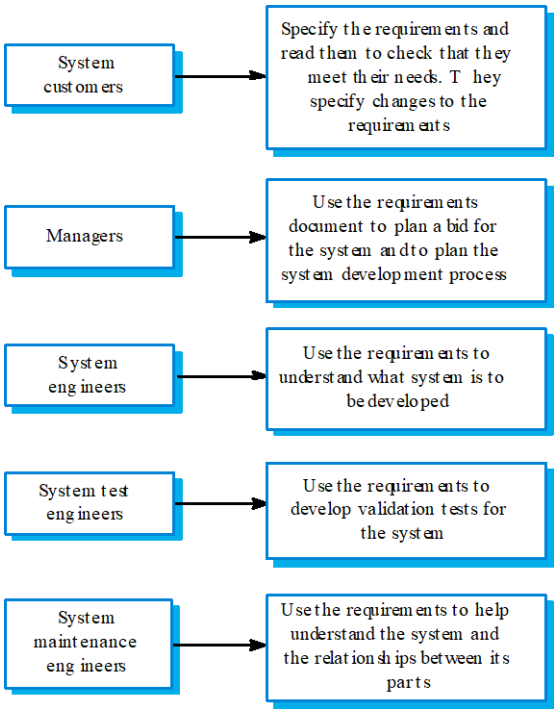

Figure 3.10 Users of Requirements Document

Figure 3.11 shows one possible organization for a requirements document that is based on an IEEE standard for requirements documents (IEEE 1998). This standard is a generic one that can be adapted to specific uses.

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.
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.

Figure 3.11 The Structure of Requirements Document

GUIDELINES FOR WRITING DOCUMENTS

- Invent a standard format and use it for all requirements.
- Use language in a consistent way. Use ***shall*** for mandatory requirements, ***should*** for desirable requirements.
- Use ***text highlighting*** to identify key parts of the requirement.
- Avoid the use of computer jargon.

LEARNING ACTIVITY 3.1

Individual Activity 3.1

Give at least 2 User Requirements and 3 System Requirements (for each user requirement) for any kind of applications of your choice under the area of emerging technologies assigned to you from the 1st group activity.

LEARNING ACTIVITY 3.2

Individual Activity 3.2

Identify at least 5 functional and 3 non-functional requirements of any application or system.

SUMMARY / KEY POINTS

- Requirements set out what the system should do and define constraints on its operation and implementation.
- Functional requirements set out services 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.
- A software requirements document is an agreed statement of the system requirements.
- The IEEE standard is a useful starting point for defining more detailed specific requirements standards.

REFERENCES

Macaulay, L.A. (1996). ***Requirements Engineering,*** *Springer, pp. 83-112*

Sommerville, Ian. Software Engineering, 10th Edition. 2016