**Exercise – 1**

**Title of the Exercise: Identifying the Requirements from Problem Statements**

# Exercise Number: 1

## Title of the Exercise : Identifying the Requirements from Problem Statements

**Date of the Exercise :**

**AIM OF THE EXPERIMENT**

Requirements identification is the first step of any software development project. Until the requirements of a client have been clearly identified, and verified, no other task (design, coding, testing) could begin. Usually business analysts having domain knowledge on the subject matter discuss with clients and decide what features are to be implemented.

In this experiment we will learn how to identify functional and non-functional requirements from a given problem statement. Functional and non-functional requirements are the primary components of a Software Requirements Specification.

**OBJECTIVES**

After completing this experiment, you will be able to:

* Identify ambiguities, inconsistencies and incompleteness from a requirements specification
* Identify and state functional requirements
* Identify and state non-functional requirements

**REQUIREMENTS:**

Sommerville defines "requirement" [1] as a specification of what should be implemented. Requirements specify how the target system should behave. It specifies what to do, but not how to do. Requirements engineering refers to the process of understanding what a customer expects from the system to be developed, and to document them in a standard and easily readable and understandable format. This documentation will serve as reference for the subsequent design, implementation and verification of the system.

It is necessary and important that before we start planning, design and implementation of the software system for our client, we are clear about its requirements. If we don't have a clear vision of what is to be developed and what all features are expected, there would be serious problems, and customer dissatisfaction as well.

**CHARACTERISTICS OF REQUIREMENTS**

Requirements gathered for any new system to be developed should exhibit the following three properties:

**Unambiguity:** There should not be any ambiguity what a system to be developed should do. For example, consider you are developing a web application for your client. The client requires that enough number of people should be able to access the application simultaneously. What's the "enough number of people"? That could mean 10 to you, but, perhaps, 100 to the client. There's an ambiguity.

**Consistency:** To illustrate this, consider the automation of a nuclear plant. Suppose one of the clients say that it the radiation level inside the plant exceeds R1, all reactors should be shut down. However, another person from the client side suggests that the threshold radiation level should be R2. Thus, there is an inconsistency between the two end users regarding what they consider as threshold level of radiation.

**Completeness:** A particular requirement for a system should specify what the system should do and also what it should not. For example, consider a software to be developed for ATM. If a customer enters an amount greater than the maximum permissible withdrawal amount, the ATM should display an error message, and it should not dispense any cash.

**CATEGORIZATION OF REQUIREMENTS:**

Based on the target audience or subject matter, requirements can be classified into different types, as stated below:

**User requirements:** They are written in natural language so that both customers can verify their requirements have been correctly identified

**System requirements:** They are written involving technical terms and/or specifications, and are meant for the development or testing teams

Requirements can be classified into two groups based on what they describe:

**Functional requirements (FRs):** These describe the functionality of a system -- how a system should react to a particular set of inputs and what should be the corresponding output.

**Non-functional requirements (NFRs):** They are not directly related what functionalities are expected from the system. However, NFRs could typically define how the system should behave under certain situations. For example, a NFR could say that the system should work with 128MB RAM. Under such condition, a NFR could be more critical than a FR.

Non-functional requirements could be further classified into different types like:

**Product requirements:** For example, a specification that the web application should use only plain HTML, and no frames

**Performance requirements:** For example, the system should remain available 24x7

**Organizational requirements:** The development process should comply to SEI CMM level 4

**FUNCTIONAL REQUIREMENTS**

**Identifying Functional Requirements**

Given a problem statement, the functional requirements could be identified by focusing on the following points:

* Identify the high-level functional requirements simply from the conceptual understanding of the problem. For example, a Library Management System, apart from anything else, should be able to issue and return books.
* Identify the cases where an end user gets some meaningful work done by using the system. For example, in a digital library a user might use the "Search Book" functionality to obtain information about the books of his interest.
* If we consider the system as a black box, there would be some inputs to it, and some output in return. This black box defines the functionalities of the system. For example, to search for a book, user gives title of the book as input and get the book details and location as the output.
* Any high level requirement identified could have different sub-requirements. For example, "Issue Book" module could behave differently for different class of users, or for a particular user who has issued the book thrice consecutively.

**PREPARING SOFTWARE REQUIREMENTS SPECIFICATIONS**

Once all possible FRs and non-FRs have been identified, which are complete, consistent, and non-ambiguous, the Software Requirements Specification (SRS) is to be prepared. IEEE provides a template [iv], also available here, which could be used for this purpose. The SRS is prepared by the service provider, and verified by its client. This document serves as a legal agreement between the client and the service provider. Once the concerned system has been developed and deployed, and a proposed feature was not found to be present in the system, the client can point this out from the SRS. Also, if after delivery, the client says a new feature is required, which was not mentioned in the SRS, the service provider can again point to the SRS. The scope of the current experiment, however, doesn't cover writing a SRS.

**CASE STUDY**

**Problem Statement:**

The VFSTR Institute has been recently setup to provide state-of-the-art research facilities in the field of Software Engineering. Apart from research scholars (students) and professors, it also includes quite a large number of employees who work on different projects undertaken by the institution.

As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would a web application (using the recent HTML 5), which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg., passwords) is stored in plain text.

**Identification of functional requirements**

The above problem statement gives a brief description of the proposed system. From the above, even without doing any deep analysis, we might easily identify some of the basic functionality of the system:

* **New user registration:** Any member of the institute who wishes to avail the facilities of the library has to register himself with the Library Information System. On successful registration, a user ID and password would be provided to the member. He has to use this credentials for any future transaction in LIS.
* **Search book:** Any member of LIS can avail this facility to check whether any particular book is present in the institute's library. A book could be searched by its:
  + Title
  + Authors name
  + Publisher's name
* **User login:** A registered user of LIS can login to the system by providing his employee ID and password as set by him while registering. After successful login, "Home" page for the user is shown from where he can access the different functionalities of LIS: search book, issue book, return book, reissue book. Any employee ID not registered with LIS cannot access the "Home" page -- a login failure message would be shown to him, and the login dialog would appear again. This same thing happens when any registered user types in his password wrong. However, if incorrect password has been provided for three time consecutively, the security question for the user (specified while registering) with an input box to answer it are also shown. If the user can answer the security question correctly, a new password would be sent to his email address. In case the user fails to answer the security question correctly, his LIS account would be blocked. He needs to contact with the administrator to make it active again.
* **Issue book:** Any member of LIS can issue a book against his account provided that:
  + The book is available in the library i.e. could be found by searching for it in LIS
  + No other member has currently issued the book
  + Current user has not issued the maximum number of books that can

If the above conditions are met, the book is issued to the member.

Note that this FR would remain incomplete if the "maximum number of books that can be issued to a member" is not defined. We assume that this number has been set to four for students and research scholars, and to ten for professors.

Once a book has been successfully issued, the user account is updated to reflect the same.

* **Return book:** A book is issued for a finite time, which we assume to be a period of 20 days. That is, a book once issued should be returned within the next 20 days by the corresponding member of LIS. After successful return of a book, the user account is updated to reflect the same.
* **Reissue book:** Any member who has issued a book might find that his requirement is not over by 20 days. In that case, he might choose to reissue the book, and get the permission to keep it for another 20 days. However, a member can reissue any book at most twice, after which he has to return it. Once a book has been successfully reissued, the user account is updated to reflect the information.

In a similar way we can list other functionality offered by the system as well. However, certain features might not be evident directly from the problem system, but which, nevertheless, are required. One such functionality is "User Verification". The LIS should be able to judge between a registered and non-registered member. Most of the functionality would be available to a registered member. The "New User Registration" would, however, be available to non-members. Moreover, an already registered user shouldn't be allowed to register himself once again.

Having identified the (major) functional requirements, we assign an identifier to each of them [v] for future reference and verification. Following table shows the list:

| **#** | **Requirement** | **Priority** |
| --- | --- | --- |
| R1 | New user registration | High |
| R2 | User Login | High |
| R3 | Search book | High |
| R4 | Issue book | High |
| R5 | Return book | High |
| R6 | Reissue book | Low |

**Table 01: Identifier and priority for software requirements**

**IDENTIFICATION OF NON-FUNCTIONAL REQUIREMENTS**

Having talked about functional requirements, let's try to identify a few non-functional requirements.

* **Performance Requirements:**
  + This system should remain accessible 24x7
  + At least 50 users should be able to access the system altogether at any given time
* **Security Requirements:**
  + This system should be accessible only within the institute LAN
  + The database of LIS should not store any password in plain text -- a hashed value has to be stored
* **Software Quality Attributes**
* **Database Requirements**
* **Design Constraints:**
  + The LIS has to be developed as a web application, which should work with Firefox 5, Internet Explorer 8, Google Chrome 12, Opera 10
  + The system should be developed using HTML 5

Once all the functional and non-functional requirements have been identified, they are documented formally in **SRS**, which then serves as a legal agreement.

**Viva Questions**

1. When is feasible study done?

Answer:

1. A Good requirement specification is one, which is

Answer:

1. When is Requirement specification is done?

Answer:

1. Functional Requirement of a system are related to?

Answer:

1. SRS Refers to?

Answer:

1. What is the main Objective behind preparing the SRS?

Answer: