Module 3

Requirements Analysis and Specification

Lesson 5

Basic concepts in Requirements Analysis and Specification

Specific Instructional Objectives

At the end of this lesson the student will be able to:

- Explain the role of a system analyst.
- Identify the important parts of SRS document.
- Identify the functional requirements from any given problem description.
- Document the functional requirements from any given problem description.
- Identify the important properties of a good SRS document.
- Identify the important problems that an organization would face if it does not develop an SRS document.
- Identify non-functional requirements from any given problem description.
- Identify the problems that an unstructured specification would create during software development.
- Represent complex conditions in the form of a decision tree.
- Represent complex conditions in the form of decision table.

Role of a system analyst

The analyst starts requirements gathering and analysis activity by collecting all information from the customer which could be used to develop the requirements of the system. He then analyzes the collected information to obtain a clear and thorough understanding of the product to be developed, with a view to removing all ambiguities and inconsistencies from the initial customer perception of the problem. The following basic questions pertaining to the project should be clearly understood by the analyst in order to obtain a good grasp of the problem:

- What is the problem?
- Why is it important to solve the problem?
- What are the possible solutions to the problem?
- What exactly are the data input to the system and what exactly are the data output by the system?
- What are the likely complexities that might arise while solving the problem?
- If there are external software or hardware with which the developed software has to interface, then what exactly would the data interchange formats with the external system be?

After the analyst has understood the exact customer requirements, he proceeds to identify and resolve the various requirements problems. The most important requirements problems that the analyst has to identify and eliminate are the problems of anomalies, inconsistencies, and incompleteness. When the analyst detects any inconsistencies, anomalies or incompleteness in the gathered requirements, he resolves them by carrying out further discussions with the end-users and the customers.

Parts of a SRS document

- The important parts of SRS document are:
    - Functional requirements of the system
    - Non-functional requirements of the system, and
    - Goals of implementation

Functional requirements:-

- The functional requirements part discusses the functionalities required from the system. The system is considered to perform a set of high-level functions {f_i}. The functional view of the system is shown in fig. 3.1. Each function f_i of the system can be considered as a transformation of a set of input data (ii) to the corresponding set of output data (o_i). The user can get some meaningful piece of work done using a high-level function.

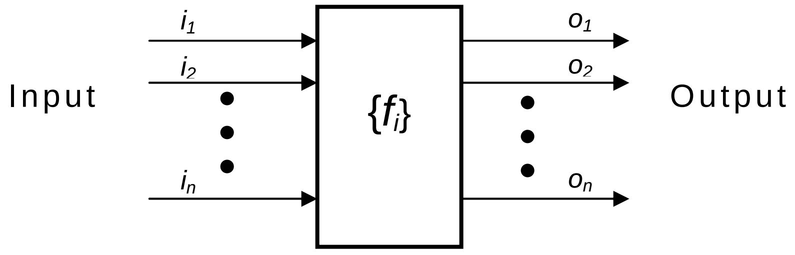

Fig. 3.1: View of a system performing a set of functions

Nonfunctional requirements:-

- Nonfunctional requirements deal with the characteristics of the system which can not be expressed as functions - such as the maintainability of the system, portability of the system, usability of the system, etc.

- Nonfunctional requirements may include:

  # reliability issues,
  # accuracy of results,
  # human - computer interface issues,
  # constraints on the system implementation, etc.

Goals of implementation:-

The goals of implementation part documents some general suggestions regarding development. These suggestions guide trade-off among design goals. The goals of implementation section might document issues such as revisions to the system functionalities that may be required in the future, 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.

Identifying functional requirements from a problem description

The high-level functional requirements often need to be identified either from an informal problem description document or from a conceptual understanding of the problem. Each high-level requirement characterizes a way of system usage by some user to perform some meaningful piece of work. There can be many types of users of a system and their requirements from the system may be very different. So, it is often useful to identify the different types of users who might use the system and then try to identify the requirements from each user's perspective.

Here we list all functions $\{f_i\}$ that the system performs. Each function f_i as shown in fig.3.2 is considered as a transformation of a set of input data to some corresponding output data.

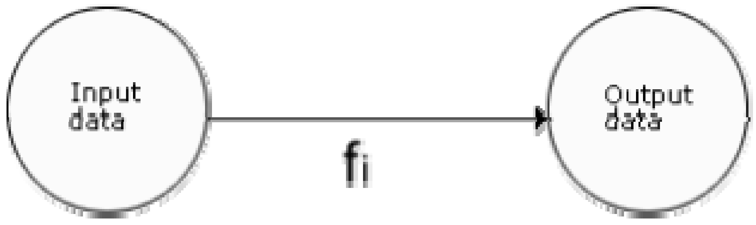

Fig. 3.2: Function f_i

Example:-
Consider the case of the library system, where **-**

F1: Search Book function (fig. 3.3)
Input: an author's name
Output: details of the author's books and the location of these books in the library

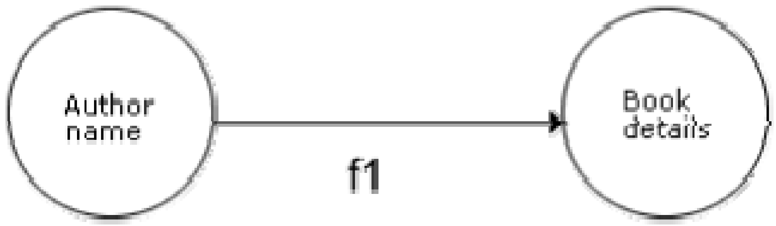

Fig. 3.3: Book Function

So the function Search Book (F1) takes the author's name and transforms it into book details.

Functional requirements actually describe a set of high-level requirements, where each high-level requirement takes some data from the user and provides some data to the user as an output. Also each high-level requirement might consist of several other functions.

Documenting functional requirements

For documenting the functional requirements, we need to specify the set of functionalities supported by the system. A function can be specified by identifying the state at which the data is to be input to the system, its input data domain, the output data domain, and the type of processing to be carried on the input data to obtain the output data. Let us first try to document the withdraw-cash function of an ATM (Automated Teller Machine) system. The withdraw-cash is a high-level requirement. It has several sub-requirements corresponding to the different user interactions. These different interaction sequences capture the different scenarios.

Example: - Withdraw Cash from ATM

R1: withdraw cash

Description: The withdraw cash function first determines the type of account that the user has and the account number from which the user wishes to withdraw cash. It checks the balance to determine whether the requested amount is available in the account. If enough balance is available, it outputs the required cash, otherwise it generates an error message.

R1.1 select withdraw amount option

Input: “withdraw amount” option

Output: user prompted to enter the account type

R1.2: select account type

Input: user option

Output: prompt to enter amount

R1.3: get required amount

Input: amount to be withdrawn in integer values greater than 100 and less than 10,000 in multiples of 100.

Output: The requested cash and printed transaction statement.

Processing: the amount is debited from the user's account if sufficient balance is available, otherwise an error message displayed.

Properties of a good SRS document

- The important properties of a good SRS document are the following:
  - **Concise.** The SRS document should be concise and at the same time unambiguous, consistent, and complete. Verbose and irrelevant descriptions reduce readability and also increase error possibilities.
  - **Structured.** It should be well-structured. A well-structured document is easy to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the customer requirements. Often, the customer requirements evolve over a period of time. Therefore, in order to make the modifications to the SRS document easy, it is important to make the document well-structured.
  - **Black-box view.** 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 external behavior of the system and not discuss the implementation issues. The SRS document should view the system to be developed as black box, and should specify the externally visible behavior of the system. For this reason, the SRS document is also called the black-box specification of a system.
  - **Conceptual integrity.** It should show conceptual integrity so that the reader can easily understand it.
  - **Response to undesired events.** It should characterize acceptable responses to undesired events. These are called system response to exceptional conditions.

- **Verifiable.** All requirements of the system as documented in the SRS document should be verifiable. This means that it should be possible to determine whether or not requirements have been met in an implementation.

Problems without a SRS document

- The important problems that an organization would face if it does not develop an SRS document are as follows:

  - Without developing the SRS document, the system would not be implemented according to customer needs.
  - Software developers would not know whether what they are developing is what exactly required by the customer.
  - Without SRS document, it will be very much difficult for the maintenance engineers to understand the functionality of the system.
  - It will be very much difficult for user document writers to write the users' manuals properly without understanding the SRS document.

Identifying non-functional requirements

Nonfunctional requirements are the characteristics of the system which can not be expressed as functions - such as the maintainability of the system, portability of the system, usability of the system, etc.

Nonfunctional requirements may include:

# reliability issues,
# performance issues,
# human - computer interface issues,
# interface with other external systems,
# security and maintainability of the system, etc.

Problems with an unstructured specification

- It would be very much difficult to understand that document.
- It would be very much difficult to modify that document.
- Conceptual integrity in that document would not be shown.
- The SRS document might be unambiguous and inconsistent.

Decision tree

A decision tree gives a graphic view of the processing logic involved in decision making and the corresponding actions taken. The edges of a decision tree represent conditions and the leaf nodes represent the actions to be performed depending on the outcome of testing the condition.

Example: -

Consider Library Membership Automation Software (LMS) where it should support the following three options**:**

- **New member**
- **Renewal**
- **Cancel membership**

New member option-

Decision: When the 'new member' option is selected, the software asks details about the member like the member's name, address, phone number etc.

Action: If proper information is entered then a membership record for the member is created and a bill is printed for the annual membership charge plus the security deposit payable.

Renewal option-

Decision: If the 'renewal' option is chosen, the LMS asks for the member's name and his membership number to check whether he is a valid member or not.

Action: If the membership is valid then membership expiry date is updated and the annual membership bill is printed, otherwise an error message is displayed.

Cancel membership option-

Decision: If the 'cancel membership' option is selected, then the software asks for member's name and his membership number.

Action: The membership is cancelled, a cheque for the balance amount due to the member is printed and finally the membership record is deleted from the database.

Decision tree representation of the above example -

The following tree (fig. 3.4) shows the graphical representation of the above example. After getting information from the user, the system makes a decision and then performs the corresponding actions.

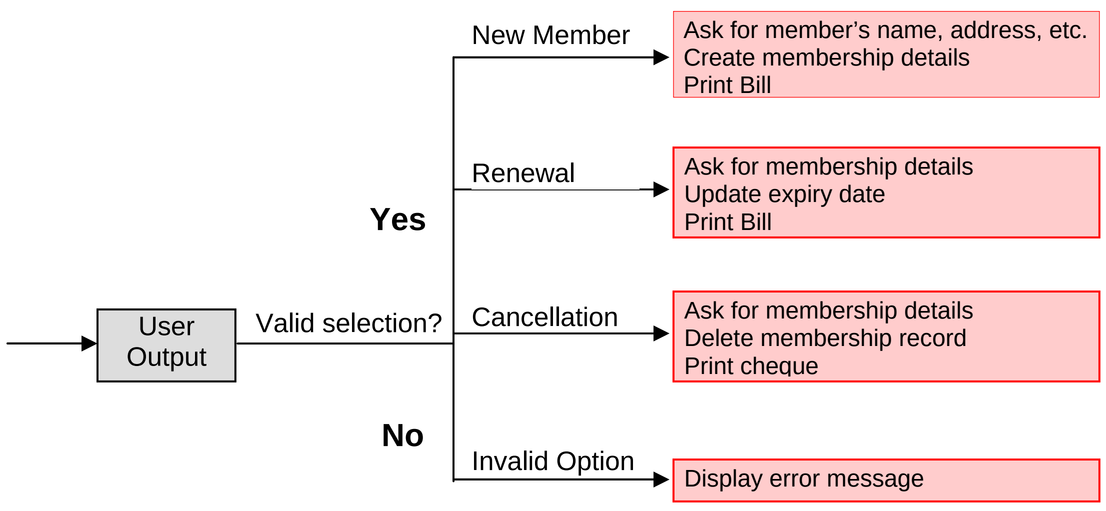

Fig. 3.4: Decision tree for LMS

Decision table

A decision table is used to represent the complex processing logic in a tabular or a matrix form. The upper rows of the table specify the variables or conditions to be evaluated. The lower rows of the table specify the actions to be taken when the corresponding conditions are satisfied. A column in a table is called a *rule*. A rule implies that if a condition is true, then the corresponding action is to be executed.

Example: -

Consider the previously discussed LMS example. The following decision table (fig. 3.5) shows how to represent the LMS problem in a tabular form. Here the table is divided into two parts, the upper part shows the conditions and the lower part shows what actions are taken. Each column of the table is a rule.

Conditions

Valid selection	No	Yes	Yes	Yes
New member	-	Yes	No	No
Renewal	-	No	Yes	No
Cancellation	-	No	No	Yes
Actions				
Display error message	x	-	-	-
Ask member's details	-	x	-	-
Build customer record	-	x	-	-
Generate bill	-	x	x	-
Ask member's name & membership number	-	-	x	x
Update expiry date	-	-	x	-
Print cheque	-	-	-	x
Delete record	-	-	-	x

Fig. 3.5: Decision table for LMS

From the above table you can easily understand that, if the valid selection condition is false then the action taken for this condition is 'display error message'. Similarly, the actions taken for other conditions can be inferred from the table.