3.1 SOFTWARE REQUIREMENTS

According to IEEE definition, a requirement is (1) a condition of capability needed by a user to solve a problem or achieve; (2) a condition or a capability that must be met or possessed by a system..., to satisfy a contract, standard, specification or other formally imposed document.....[IEE87]. The software requirements deal with the requirements of the proposed system and produce a document at the end of the requirements phase of software development cycle. And this document is known as the Software Requirements Specification (SRS). SRS completely describes what the proposed software system should do without describing how the software will do it.

In Systems Engineering and Software Engineering, Requirements Analysis encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users. Systematic requirements analysis is also known as *requirements engineering*. It is sometimes referred to loosely by names such as *requirements gathering, requirements capture,* or *requirements specification*. The term *requirements analysis* can also be applied specifically to the analysis proper (as opposed to elicitation or documentation of the requirements, for instance).

Requirements analysis is critical to the success of a development project.

Requirements must be actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design. IEEE defines a requirement as

- A condition or capability needed by a user to solve a problem or achieve an objective
- A condition or capability that must be met or possessed by a system to satisfy a contract, standard, specification or other formally imposed document

In software requirements we are dealing with the requirements of the proposed system, that is the capabilities that the system, which is yet to be developed, should have. It is because we are dealing with specifying a system that does not exist in any form (the manual form of existence does not generally have the same capability as the eventual automated system) that the problem of requirements becomes complicated. Regardless of how the requirements phase proceeds, it ultimately ends with the Software Requirement Specification (SRS). Generally, The SRS is a document that completely describes what the proposed softwares should do without describing how the software will do it. The basic goal of the Requirement Phase is to Produce the SRS, which describe the complex external behaviour of the proposed software. Requirement analysis involves number of suggestions such as:

- Understanding the Problem
- Record the origin of and the reason for every requirement
- Use multiple views of requirements
- Set the priorities of requirements
- Functionally decompose the requirements
- Try to eliminate ambiguity
- Reduce the complexities

Analyst should consider the above suggestions for good quality requirement analysis. Analyst collects detailed information regarding the project. During the detailed requirement analysis, analyst usually carries out two activities, the first, requirements/information gathering and second is analysis of gathered requirements or information. All software applications, collectively called as data processing software, are built to process data to transform data from one form to another, *i.e.*, to accept input, manipulate it in some way, and produce output. We can say requirement analysis is playing very important role for data processing. If software processes wrong data, it will produce wrong results or incorrect information.

3.1.1 Analysis Principles

There are a number of analysis principles for requirements analysis. We can use any analysis method but all analysis methods are related to the following set of operational principles.

- The information domain of the problem should be understood
- The functions that the software is to perform must be defined
- The behaviour of the software must be represented
- The analysis should more from essential information toward detail information
- The analysis process should use the multiple views of requirements

3.1.2 Analyzing a problem

The basic purpose of problem analysis is to understand the problem and its constraints. Its main aim is to understand the needs of the clients and the users, what they want from the system. Analysis leads to the actual specification. It involves interviewing the clients and the end users. The existing documents, clients and end users become the main source of information for the analysts. There are a number of methods to problem analysis. And we will discuss a few among them like informal approach, structured analysis, prototyping.

Informal approach

This approach has no defined methodology. The information about the system is obtained by interaction with the client, end users, study of the existing system etc. It uses conceptual modeling. That is, the problem and the system model are built in the minds of the analysts and it is directly translated to SRS. Once the initial draft of the SRS is ready, it may be used in further meetings.

Structured analysis

This method focuses on the functions performed in the problem domain and the data input and output by these functions. It is a top-down refinement process; it helps an analyst to decide upon the type of information to be obtained at various stages of analysis. This technique mainly depends on two types of data representation methods, the data flow diagram (DFD) and data dictionary.

3.4 SOFTWARE REQUIREMENT SPECIFICATION

A Software Requirements Specification (SRS) is a complete description of the behaviour of the system to be developed. It includes a set of use cases that describe all of the interactions that the users will have with the software. Use cases are also known as functional requirements. In addition to use cases, the SRS also contains nonfunctional (or supplementary) requirements. Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints).

A SRS document should clearly document the following:

- Functional requirements of the system
- External interfaces of the system
- Non-functional requirements of the system
- Design constraints

A good SRS document consists the various qualities, such as it should be concise, unambiguous, consistent and complete.

Following background information should be considered while writing an SRS :

- Nature of the SRS
- Environment of the SRS
- Characteristics of the SRS
- Joint preparation of the SRS
- SRS evolution
- Embedding design in the SRS
- Embedding project requirements in the SRS

SRS has a specific role to play in the software development process, the SRS writer should be careful not to go beyond the bounds of that role. This meant that the SRS :

(i) Should correctly define all the software requirements. Software requirements may exist because of the nature of the task to be solved or because of special characteristics of the project.

(ii) Should not describe any design or implementation details. These should be described in the design stage of the project.

(iii) Should not impose additional constraints on the software. These are properly specified in other documents such as a software quality assurance plan.

Therefore, a properly written SRS limits the range of valid designs, but does not specify any particular design.

3.4.1 Characteristics of a Good SRS

To properly satisfy the basic goals, an SRS should have certain properties and should contain different types of requirements. A good SRS is :

(i) Correct

(ii) Complete

(iii) Unambiguous

(iv) Verifiable

(v) Consistent

(vi) Ranked for importance and/or stability

(vii) Modifiable

(viii) Traceable

An SRS is correct if every requirement included in the SRS represents something required in the final system. An SRS is complete if everything the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS. Correctness and Completeness go hand-in -hand, whereas correctness ensures that what is specified is done correctly, completeness ensures that everything is indeed specified. Correctness is an easier property to establish than completeness as it basically involves examining each requirement to make sure it represents the user requirement.

An SRS is unambiguous if and only if every requirement stated has one and only one interpretation. Requirements are often written in natural language, which are inherently ambiguous. If the requirements are specified in a natural language, the SRS writer has to be especially careful to ensure that there are no ambiguities. One way to avoid ambiguities is to use some formal requirements specification language. The major disadvantage of using formal languages is the large effort required to write an SRS, the high cost of doing so, and the increased difficulty reading and understanding formally stated requirements.

An SRS is verifiable if and only if every stated requirement is verifiable. A requirement is verifiable if there exist some cost effective process that can check whether the final software meets the requirements. This implies that the requirements should have as little subjectivity as possible because subjective requirements are difficult to verify. Unambiguity is essential for verifiability.

An SRS is consistent if there is no requirement that conflicts with another. Terminology can cause inconsistencies; for example, different requirements may use different terms to refer to the same object. There may be logical or temporal conflict between requirements causing inconsistencies. This occurs if the SRS contains two or more requirements whose logical or temporal characteristics can not be satisfied together by any software system.