SEMCOM

Faculty Name: Dr. Jay Nanavati
Sem: 5 (TYBCA)
Subject: Software Engineering
Unit-2

SOFTWARE REQUIREMENT SPECIFICATION (SRS)

Requirement

IEEE defines a requirement as:
"A condition or capability needed by a user to solve a problem or achieve an objective."

"A condition or a capability that must be met or possessed by system to satisfy a contract, standard, specification or other formally imposed document."

Initially, the software requirement was not given much importance. Coding and Design were given more importance. The developers understood the problem when it was explained to them informally.
As systems grew more complex, it was not possible to understand the goals of systems completely. More thorough requirement analysis was required. For such large software systems requirement analysis was difficult and error prone.

The software project is initiated by client's need. The requirement analyst identifies requirements by talking to people in client organization. Thus, identifying requirements involves specifying what some people have in their minds.

The requirement analysis phase takes these ideas as an input and translates them in to formal document, which is an output of this phase.

Software Requirement Specification is a document that completely describes what the proposed software should do without describing how the software will do it.

Need/Role/Benefits of SRS

1. An SRS establishes the basis for the agreement between the client and supplier on what the software will do.

Justification:
There are three major parties involved in development of new system: the client, the user and developer. The client usually does not understand the software and the software development and the developer do not understand the client's problem and application area. This causes communication gap between the parties involved in the development project. The basic purpose of SRS is to bridge this communication gap.
SRS is a medium through which client and user needs are accurately specified.
SRS forms the basis for software development. A good SRS should satisfy all the parties. Through SRS client clearly describes what it expects from supplier and the supplier clearly understands what capabilities to build in the software.
Thus client and supplier agree on what is required and what is to be developed through an SRS.

2. An SRS provides a reference for validation of final product.

Justification:
An SRS is document that specifies the requirements of users and client.
An SRS helps client to determine if software meets the requirements specified by the client and the user.
When software is developed completely we need to determine if the software being delivered is what was ordered.
Without proper SRS, the client cannot determine whether software is developed as per the requirements. Even the supplier cannot convince the client that all requirements are fulfilled.
SRS acts as proof for users and clients to validate the product against the requirements.

3. A high-quality SRS is prerequisite to high-quality software.

Justification:
For a high-quality SRS a high-quality SRS is a must.
Many errors are made during requirement analysis phase.
An error in SRS is most likely to appear in the final system.

If SRS document specifies the wrong system then even correct implementation of SRS will lead to a system that will not satisfy the client.
Thus if we want a high-quality software we must begin with high-quality SRS.

4. A high-quality SRS reduces development cost.
Justification:
The quality of SRS has an impact on cost of project.
Errors can exist in SRS.
The cost of fixing the errors increases almost exponentially as time progresses.
If an error in requirement phase is detected and removed after system has been developed, it can cost up to 100 times more than removing it during the requirements phase itself.
From the relative cost of fixing errors as reported in a multi-company study, the approximate average cost of fixing requirements errors depending on phase is as shown below:

Phase	Cost (persons-hours)
Requirements Analysis	2
Design	5
Coding	15
Acceptance Test	50
Operations &Maintenance	150

As shown in table the requirement analysis phase involves cost of engaging 2 persons for error fixing and this cost increases subsequently as we move on to next phase and it is maximum during Operations and Maintenance phase.

There is tremendous reduction in project cost if errors in SRS solved during requirement phase only.

Problem Analysis

The aim of problem analysis is to obtain clear understanding of the needs of the clients and users, what a software should do and what are the constraints the must be considered.

Issues involved in Analysis:

1. To obtain the necessary information (sources and methods)
2. To organize the information obtained (completeness and consistency)
3. To resolve contradictions (consistent final specifications)
4. To avoid internal design (focus on what)

Activities of Problem Analysis:

1. To collect information regarding existing system and requirements of users and clients
2. To organize the information so that it can be evaluated for completeness and consistency
3. To resolve the contradictions that may exist in the information from different parties.

Structuring Information:
In problem analysis information is collected using interviews, questionnaires, documentation, etc. This information must be complete and consistent. To have such complete and consistent information we need to structure (organize) the information. A good structuring of information will guide the analyst and help him determine the areas of incompleteness. To have consistent information the analyst must resolve the conflict among the people. Identifying conflicts requires good structuring of information.

Techniques (Principles) of Analysis/ Structuring Information:

1. Divide and Conquer
In this technique the analyst partition the problem in to subproblems and try understand each subproblem and its relationship to other subproblems in an effort to understand the total problem.
This partitioning is done with respect to:
1.Objects
2.Functions

An object is an entity in real world that has clearly defined boundaries and an independent existence.
A function is a task, service, process or a mathematical function, or an activity that is now being performed in the real world or has to be performed by the system that will be built to solve to solve real world problem.

2. Projection

In projection, a system is defined from multiple points of view. While using projection different viewpoints of system are defined and the system is analyzed from these different perspectives using an object-based or a function-based approach. The projections obtained are combined to form analysis for the complete system.
For example, projecting a three-dimensional object on three different two-dimensional planes is a similar process.
The advantage of using projection is that trying to analyze the system from global viewpoint is difficult and error-prone and one likely to forget some features of the system. Analyzing the system from different perspectives is easier, as it limits and focuses the scope of study.
For example, an operating system can be analyzed from different perspectives of a user, a system programmer or a system administrator.

Approaches to Problem Analysis

There are three basic approaches to Problem analysis:

1. **Informal approach**
   - ✓ No defined methodology is used.
   - ✓ The information is obtained by
     - interaction with the clients, end users
     - questionnaires
     - study of existing documents
     - brainstorming
   - ✓ Widely used, gives information on all aspects of the system.

2. **Structued Analysis**
   - ✓ The problem is viewed as a set of functions to perform.
   - ✓ A top-down approach.
   - ✓ Uses DFDs & Data dictionary.

3. **Object-oriented Modeling**
   - ✓ The problem is viewed as a set of objects.
   - ✓ A bottom-up approach.
   - ✓ Uses an object model.

Characteristics of SRS

1. Correct and Complete:

An SRS is correct if every requirement included in SRS represents something required in the final system.
An SRS is complete if everything 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.
Correctness ensures that what is specified is done correctly.
Completeness ensures that everything is indeed specified.
Correctness is easier to establish because it involves examining each requirement to make sure that it represents the user requirement.
Completeness is more difficult to establish because one has to detect the absence of specifications.

2. Unambiguous:

An SRS is unambiguous if and only if every requirement stated has one and only one interpretation.
Requirements are often written in natural languages, which are very ambiguous. One way to avoid ambiguity is to use formal languages to write SRS. The disadvantage of this formal language is large effort is required write an SRS, high cost is involved, increased difficulty in reading and understanding formally stated requirements.

3. Verifiable: An SRS is verifiable if and only if the requirement is verifiable. A requirement is verifiable if there exist some cost-effective process to check whether the final software meets that requirement. Unambiguity is essential for verifiability. As verification of requirements is done through review it implies that SRS should be understandable.

4. Consistent:

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 for the same object. Logical or Temporal conflict may also cause inconsistencies. This may occur if there is one requirement, which may conflict with each other. For example one requirement states that event a occurs before event b. But requirement states that event b occurs before a. Thus both requirements conflict with each other.

5. Ranked for Importance/Stability:

All the requirements specified in SRS are not of equal importance. So we give them rank according to their importance.

The requirements that are very much important are ranked as critical. The requirements that are not very much important are ranked as important. The requirements that are not important are ranked as desirable.

Stability reflects the chances of changes. The requirements, which do not change, are ranked as stable. The requirements that change with time are ranked as not stable.

6. Modifiable:

An SRS is modifiable if we can change the structure and style of SRS while preserving the completeness. Presence of redundancy is major problem while doing modification because it leads to many errors. For example, suppose a requirement is stated at two places and later on it required to be changed. If we made change only at one place and forgot to make change at another place it results in error.

7. Traceable:

An SRS is traceable if each requirement stated can be used as a reference for developing software. Forward traceability means that each requirement stated in should be traceable to design and code. Backward traceability means that design and code should be traceable to requirements.

Components of SRS

1.Functional Requirements:

Functional requirements specify which outputs should be produced from the given inputs and describe relationship between them. For each functional requirement a detailed description of inputs and their source is specified. All the operations to be performed on input data to obtain the output should be specified. The algorithms that are not part of existing system are not specified. The important thing that must be specified is the system behaviour in abnormal situations. For example, invalid input or error during computation.

The behaviour of system where input is valid but normal operations cannot be done should also be specified. For example, in an airline reservation system a reservation cannot be made in for valid passengers if the airplane is already booked.

2.Performance Requirements:

Performance Requirements specifies performance constraints on the software system. All requirements relating to performance characteristics of the system must be specified.

There are two types of performance requirements:

1.Static:

The requirements that do not impose constraint on the execution behaviour of the system are called Static requirements. These requirements include number of terminals to be supported, number of users to be supported, the number of files that are to be processed and their sizes. They are also called capacity requirements.

2.Dynamic:

The requirements that specify constraints on the execution of the behaviour of system are called Dynamic requirements. They include response time and throughput.

Response time is the time expected for completion of an operation under specified circumstances.

Throughput is the expected number of operations that can be performed in a unit time.

All requirements should be stated in measurable terms.

Requirements such as “response time should be good” are not desirable. Instead we must state that “response time for command x should be less than one second 90% of the times”.

3.Design constraints:

Design constraints specify the limitations that exist in client’s environment. An SRS should identify and specify all constraints described as follows:

1.Standards Compliance:

This specifies the requirements for the standards that the system must follow. The standard includes report format and accounting procedures.

2.Hardware limitations:

This specifies the hardware on which the software is going to operate. Hardware limitations include the type of machines to be used, operating system available on the system, languages supported, and limits on primary and secondary storage.

3.Reliability and Fault Tolerance:

Fault tolerance requirements specify how a system will behave when some fault occurs. These requirements place major constraints and make system more complex and expensive. Recovery requirements specify what a system should do is failure occurs. Reliability requirements are very important for critical applications.

4.Security:

Security requirements are very significant in defense systems and many database systems. Security requirements place restrictions on use of certain commands, control access to data, provide different kinds of access requirements for different people, require use of passwords, and cryptography techniques and maintain a log of activities of the system.

4.External Interface Requirements:

External Interface Requirements specifies all the possible interactions of the software with people, hardware and other software.

For user interface, the characteristics of each user interface should be specified. A preliminary user interface must be created with all user commands, screen formats, an explanation of how system will appear to the user, feedback and error messages.

Hardware interface requirements specify all the characteristics of interface between the software product and the hardware components.

We must specify the interface with other software the system will use or that will use the system. This includes the interface with operating system and other applications.

Specification Languages

Languages which are used to write SRS are called Specification Languages.

1.Structured English

Natural Languages have been widely used for specifying requirements. The major advantage of using natural languages is that both client and supplier understand the language. Initially when software systems were small requirements were conveyed verbally using natural language for expression. Later as the software requirements grew more complex they were specified in written form rather than orally.

Disadvantage of Natural Language:

The requirements written in natural language will be imprecise and ambiguous. Efforts to be more precise and complete result in voluminous requirement specification.

To overcome these drawbacks we make use of formal language. To reduce some drawbacks natural language is used in structured fashion. Choosing English as natural language and using it in structured fashion is called Structured English. In Structured English requirements are broken down into sections and paragraphs. Each paragraph is then broken into subparagraphs. Many organizations also specify strict uses of words like “shall”, “perhaps” and “should” and try to restrict the use of common phrases in order to improve the precision and reduce the verbosity and ambiguity.

2.Regular Expressions

Requirement expressions can be used to specify structure of symbol strings. String specification is useful for specifying such things as input data, command sequence, and contents of message. Regular expressions can be considered as grammar for specifying the valid sequences in a language and can be automatically processed.

There are few basic constructs allowed in a regular expressions:

Atoms:

Atoms are the basic symbols or the alphabet of the language.

e. g. a, e, r, etc.

Composition:

This construct is formed by concatenating two regular expressions. For regular expressions r1 and r2, concatenation is expressed by (r1, r2).

e. g. Name=(FirstName, LastName)

Alternation:

This construct specifies either /or relationship. For r1 and r2, alternation is specified by (r1|r2) and denotes the union of the sets of strings specified by r1 and r2.

Number=digit digit where digit=(0|1|2…|9)

Closure:

This constructs specifies repeated occurrences of a regular expression. For a regular expression r, closure is specified by r*, which means that string denoted by r is concatenated zero or more times.

e. g. emp_file=(number name)*

3.Decision Table:

Decision table provide a mechanism for specifying complex decision. This formal, table-notation can be used check for qualities like completeness and lack of ambiguity.

A decision table has two parts. The top part lists the different conditions and the bottom parts lists different conditions. The table specifies under what combinations of conditions what actions are to be performed. If there are N conditions specified a complete decision table will have 2^N different combinations of conditions listed for which actions must be specified. These combinations are called decision rules. However not all the combinations are feasible.

We take an example of system that grants loan.

Different conditions that must be checked are:

1. Customer has bank account.
2. Customer has no due amount.
3. Customer has management approval.

The possible actions are:

1. Grant Loan
2. Reject Loan

Different possible combinations of conditions are:

1. Customer has account no dues.
2. Customer has account and some dues from previous loan and he has management approval.
3. Customer has account and some dues are there and ha has no management approval.
4. Customer has no account.

The decision table for this system is given as below:

	1	2	3	4
C1	Y	Y	Y	N
C2	Y	N	N	
C3	Y	N		
A1	X	X		
A2			X	X

For condition Y means Yes or True, N means No or False and a blank means that it can be either true or false. If an action is to be taken for particular combination of the combinations, it is shown by an X for that action. If there is no mark for an action for particular combinations of conditions, it means that the action is not to be performed.

Structure of SRS

The SRS document should be clear and concise. For this it is necessary to organize the document as sections and subsections. The reason for structuring the SRS document is that with an available standard, each SRS will fit a certain pattern that will make it easier for others to understand. It helps ensure that the analyst does not forget some major property.

The structure of SRS proposed by IEEE:

IEEE recognizes the fact that different projects may require their requirements to be organized differently. It provides different ways of structuring the SRS. The first two sections of SRS are the same in all of them. The general structure of SRS is given below:

1. Introduction
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Definitions, Acronyms and Abbreviations.
    - 1.4 References
    - 1.5 Overview
2. Overall Description

Product Perspective
Product Functions
User Characteristics
General Constraints
Assumptions and Dependencies.

3.Specific Requirements

External Interface Requirements
User Interfaces
Hardware Interfaces
Software Interfaces
Communication Interfaces
Functional Requirements
Mode 1
3.1.1.1Functional Requirements 1.1
.
.
3.2.1.n Functional Requirements 1.n
.
.
3.2.m Mode m
3.2.m.1 Functional Requirements m.1
.
.
3.2.m.n Functional Requirement m.n
Performance Requirements
Design constraints
Attributes
Other Requirements

Validating SRS

The development of software starts with developing SRS document. An SRS document can be used to determine whether delivered software is the one which was ordered That is whether the software satisfies the all the requirements stated in SRS. So it is necessary that SRS does not have errors and it should clearly specify the client's requirements correctly.

The basic objective of requirements validation activity is to ensure that the SRS reflects the actual requirements accurately and clearly. A related objective is to check that SRS document is of good quality.

There four types of errors that may occur in SRS:

1.Omission:

When a requirement is not included in SRS then such a error is called Omission.

2.Inconsistency:

When a requirement is contradictory to some other requirement or stated requirements are not compatible with actual requirements then such error is called inconsistency.

3.Incorrect Fact:

When a fact recorded in SRS is incorrect then such an error is called Incorrect fact.

4.Ambiguity:

When requirement stated in SRS have multiple meanings and its interpretation is not unique then such an error is called Ambiguity.

Methods of Validation:

1.Requirement Review:

Requirement review is a review by a group of people to find errors and point out matters of concern in the requirement specification of a system.

The review group should include the author of requirements document, someone who understands the needs of the client, a person of the design team, the person responsible for maintaining the requirements document and some people not directly involved with product development like software engineer.

The primary goal of the review process is to reveal any errors in the requirements. It is also used to consider factors affecting the quality, such as testability and readability. During review one of the jobs of the reviewers is to uncover the requirements that are subjective and too difficult to define the criteria for testing that requirement.

Checklists are frequently used in reviews to focus the review effort and ensure that no major source of errors is overlooked by the reviewers.

2.Reading:

The goal in reading is to have someone other than the author of the requirements read the requirements specification document to identify the potential problems. By having the requirements being read by the another person who may have different interpretation of requirements the errors of the misinterpretations and ambiguity is identified.

Reading is effective only if the reader takes his job seriously and reads the requirements carefully.

3.Constructing Scenarios:

Scenarios describe different situations of how system will work once it is operational. The most common area for constructing scenarios is system-user interaction. Constructing scenarios is good for clarifying misunderstandings in human-computer interaction area.

4.Prototyping:

In this method a prototype is developed to verify the requirements. It is very useful in verifying feasibility of some of the requirements.

For example, if the prototype has most of the user interfaces and they have been approved after use by client and users, then the user interface as specified by the prototype can be considered validated.

5.Automated Cross Referencing:

Automated Cross Referencing uses processors to verify some of the properties of the requirements. Any automated processing of the requirements is possible if the requirements are written in formal language or language specifically designed for the machine processing. These tools check internal consistency and completeness.

Software Project Planning

Why is planning required?

- A large software project may involve
  - ✓ hundreds of people
  - ✓ large no. of resources
  - ✓ considerable time.
- So, project management is must.
- Without planning, management is not possible.
- So, planning is also must.

Goals of planning

- ✓ To identify the activities which must be done to complete the project successfully.
- ✓ To plan the schedule.
- ✓ To allocate resources.

- Input : The SRS
- Output : The Project Plan (a document)

- **The Project Plan includes**
  - ✓ Cost estimation (Cost)
  - ✓ Schedule & milestones (Time)
  - ✓ Personnel plan (Human Resources)
  - ✓ Software Quality Assurance plans (Quality)
  - ✓ Configuration Management plans (Change)
  - ✓ Project Monitoring Plans (Monitoring)
  - ✓ Risk Management (Risk)

Cost Estimation

- **What is cost estimation?**

  To know how much it will cost for a given set of requirements & overall development of the software.

- **Why is it required?**
  1) To enable the client/the developer to perform cost-benefit analysis.
  2) For project monitoring & control.
  3) For preparing the contract.

- Cost in case of a software project involves cost of
  - ✓ Human resources (major contributor) (PM)
  - ✓ software
  - ✓ hardware

Most cost estimation methods are based on PM.

- Uncertainties in cost estimation
  - ✓ Cost estimation can be performed at any stage during SDLC.
  - ✓ Accuracy of estimate depends on amount of reliable information of the final product.
  - ✓ Thus, cost can be accurately determined after the software is developed.
  - ✓ The other extreme is when feasibility study/ RA is performed.
  - ✓ Cost can be accurately specified after the design is complete.

- Size estimation is easier than cost estimation.
- Why so?
  For estimating the size , the system is divided into components.
  Size of the components can be estimated quite accurately.
  Size of all the components can be added up to estimate the overall size of the software.
  This is NOT true for cost. (integration & other costs are involved)

COCOMO Model

- COnstructive COst Model - is a method for calculating efforts involved in a software project, proposed by Boehm.
- The effort estimate includes development, management & support tasks but does not include the cost of secretarial & other staff.
- Size is considered as the most important factor.

- Three basic steps:

1) Obtain an initial estimate of the development effort from KDLOC (i.e. size) [$E_i = a * (KDLOC)^b$]
2) Determine a set of 15 multiplying factors from different attributes of the project. [EAF]
3) Adjust the effort estimate by multiplying the initial estimate with all multiplying factors.
   [$E = EAF * E_i$]

- Step-1 Determine E_i . [$E_i = a * (KDLOC)^b$]
- a and b are known from the table given below:
-

Project type	**a**	**b**
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

- KDLOC is indirectly known.

- Step-2 Determine Efforts Adjustment Factor. (Multiplication of cost drivers)

Cost Driver	Very low	Low	Nominal	High	Very High
Required reliability	.75	.88	1.0	1.15	1.4
Database size		.94	1.0	1.08	1.16
Product complexity	.7	.85	1.0	1.15	1.3
Execution time constraint			1.0	1.11	1.3
Memory constraint			1.0	1.06	1.21
Analyst capability	1.46	1.19	1.0	.86	.71
Application experience	1.29	1.13	1.0	.91	.82
Programmer capability	1.42	1.17	1.0	.86	.70
Use of software tools	1.24	1.10	1.0	.91	.83
Development schedule	1.23	1.08	1.0	1.04	1.1

- Step-3 Determine E. [$E = EAF * E_i$]

- In COCOMO, effort for a phase is considered a defined percentage of the overall effort.
- 3 models for varying complexity: basic, intermediate and detailed.

Software Quality Assurance Plans (SQAP)

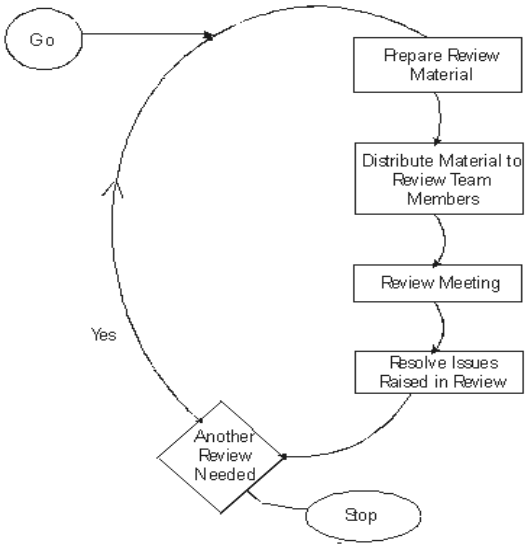
- Some quality control activities must be performed throughout the SDLC.
- The purpose of SQAP is to specify …

✓ all the work products must be produced during the project,
✓ activities that must be performed for checking the quality of each of the work products and
✓ the tools & methods that may be used for SQA activities.

- Methods used for QA

1. Verification & Validation (V&V)
2. Inspection & Reviews

- Verification & Validation (V&V)
  - Verification & validation have different meanings.
  - Verification – whether or not the products of the given phase fulfill the specifications established during previous phase.
  - Verification includes : Inspection & Reviews.
  - Validation is the process of evaluating software at the end of software development to ensure compliance with the software requirements.
  - Validation includes : Testing
  - Input : SRS
  - Output : The V&V Plan
  - The V&V Plan contains:
    - Different V&V tasks & their contribution for different phases
    - Methods to be used
    - Responsibilities & mile-stones for each of these activities
    - Input & output for each V&V task
    - Criteria for evaluating the output
- Inspections & Reviews
  - IEEE definition: " A formal evaluation technique in which software requirements, design or code are examined by a person or a group other than the author to detect faults, violations of development standards and other problems."
  - Inspection followed by meeting.
  - An inspection can be held for any technical product.
  - Reasons for performing inspections:
    1) Defect removal
    2) Productivity increase
    3) Provide information for project monitoring.
  - The most common work products inspected …
  - Provides guidelines for future.



Project Planning & Project Monitoring

(Pls. refer PPTs)