**CHAPTER # 04**

**REQUIREMENTS DOCUMENTATION**

* 1. **Requirements Documentation : An Overview**

**Documentation is the descriptive information** (e-g **hardcopy manuals, online help files, web sites) that portrays the use and/or operation of the system**. Requirements documentation is essential as this is the only **lasting impression of the requirements elicitation and analysis process** that is available to the development team, the users and the customer as the system progresses to the later parts of the software lifecycle. The requirements document usually called the software requirements specification (**SRS**) forms the basis for development as well as testing activities. The SRS is typically used by the following entities:

* **Customers and users for understanding what they are expected to get.**
* **Project Managers to estimate and plan the project to deliver the system.**
* **Designers and programmers to know what to build.**
* **Testers to prepare for testing activities.**
* **Maintenance team for understanding the system that they will maintain.**
* **Trainers to prepare the training material for training the end users.**
* **Users to understand the proposed system and prepare for the change over.**

The software requirements specification (SRS) is produced **at the end of the elicitation and analysis tasks.** The SRS may be documented in a combination of natural language textual narrative, interleaved with **graphical models, schematic and formal specification languages**. The SRS needs to include **all requirements and the authors should not assumed that certain requirements are “obvious”.**

The objective of documentation is to allow an independent person to review the system and to **understand the basis for its coding**. Software documentation is intended to help scientists, engineers and programmers who develop or review software to deal with the complexity of software products.

A written Software Requirements Document is the **principal way to record software requirements. T**he project manager and the software developer work together to record the agreements made as requirements are defined. A software prototype, accompanied by supporting text, may also function as a requirements document. The documentation of any system **describes the purpose and implementation requirements of the system.**

**The Software Requirements Document (SRD) defines the following:**

* **Defines the planned features and functions of the software product.**
* **Describes other qualities that the software must have, such as usability attributes or regulatory compliance.**
* **Clearly defines and prioritizes the customer requirements and software capabilities and features that the engineering team must deliver in the final software product.**
* **Clearly defines the target market: business problem, user requirements, and market opportunity.**
* **Identifies existing solutions and competitive products.**
* **Compares and contrasts the features, costs and capabilities of the product under development with existing commercial products.**
* **Identifies long-term plans for future releases and features.**
* **Identifies maintenance and support costs.** 
  1. **Characteristics Of SRS Document**

The requirements specification document typically called as the software requirements specification (SRS) should address the issues related to functionality external interfaces, non-functional requirements as well as design constraints. A “good” SRS document has the following characteristics:

* Completeness
* Clarity
* Correctness
* Consistency
* Modifiability
* Traceability
* Feasibility
* Testability
  + 1. **Completeness**

This is the one of the most difficult to spot. If requirements elicitation doesn’t involve representatives of different types of users the complete set of requirements may not be elicited. **Missing requirements are difficult to track since they are not visible.** We can aim for high degree of completeness of requirements by ensuring the following:

* **Elicitation of the requirements from all the “Stakeholders”.**
* **Focusing on user tasks, problems, bottlenecks and improvements required; rather than the system functionality.**
* **Ranking or prioritizing each requirement** (functional as well as non-functional) for importance. Prioritizing requirements guide customers to get more careful considerations to each requirements, and this brings to surface any hidden assumptions that they may have made.
* **Marking areas where requirements are not known as “To Be Determined” (TBD),** along with a description of **why the requirement is still TBD** and a description of what needs to be done to eliminate the TBD (including who and when).
* **Resolving all TBDs before** the design phase.

To be considered “Complete” the SRS document should contain **all functional and non-functional requirements and features that are expected in the system**. This includes the identification of all realizable classes of valid and invalid input data and the specification of the expected system responses to these input data.

* + 1. **Clarity**

The documented requirements should need to only a single interpretation, independent of the person or the time when the interpretation is done. The SRS need to be **unambiguous to the authors, the users, other reviewers as well as the developers and testers who** will use the document. Focusing on clarity only for developers and testers may prove insufficient if such clarity reduces the understandability of the SRS for the users. Here are some issues related to clarity:

* Since natural language is imprecise and ambiguous in nature any SRS written in natural language needs to reviewed by an independent entity (independent of the analyst, developers, and users) to identify ambiguous use of the natural language.
* One way of avoiding ambiguity is to write the SRS in a requirement specification language, which has a processor that detects many of the **semantic, syntactic and lexical errors**. How ever such languages take time to learn and many users find them unintelligible.
* The requirements may also be expressed using requirements analysis and modeling techniques (dataflow diagrams, use-cases, etc). These techniques improve the precision of the requirements however the initial effort in converting the requirements into the models may be disproportionately high for small systems.
  + 1. **Correctness**

The SRS can be considered as correct **if every requirement stated in the SRS is required in proposed system. There are no real tools or procedures that ensure correctness.** If there are any preceding documents then the “requirements” from those earlier documents need to be traced to the SRS. In addition all stakeholders specially users need to review the SRS and confirm that the requirements stated in the SRS correctly reflect their needs.

* + 1. **Consistency**

Requirements at all levels must be consistent with each other. **Any conflict between requirements within the SRS must be identified and resolved.** The types of conflicts that are likely to be found in an SRS are:

* Characteristics (format, details) of a real world entity interacting with the system may be conflicting.
* There may be a **logical or temporal conflict** between two specified items.
* The **terminology used for some entities/events may be** different.
  + 1. **Modifiability**

**The SRS needs to be documented in such a manner that a history of changes can be contained in the document. It will also be necessary to be able to highlight and trace the changes to the requirements** as we progress through the project. Certain good practices that can lead to high modifiability are:

* **Minimal Redundancy** Though redundancy **itself is not an error, it can lead to inconsistencies when changes are incorporated.**
* **Labeling** If requirements are labeled (or numbered), **it is easy to pinpoint the requirements that have changed.** This includes labels and references to figures, tables, diagrams and appendices in the SRS.

According to IEEE standard 830-1993 an SRS is said to be modifiable if “**its structure and style are such that any changes to the requirements can be maid easily, completely and consistently while retaining its structure and style”.**

* + 1. **Traceability**

**Labeling or numbering of requirements provides a mechanism to trace each requirement through the design, into the test plans**, test cases and source code. Fine grained, structured and precise statements are much more preferable to large, narrative paragraphs. Expected traceability is of two types:

* **Backward traceability** to the documentation and other work products created prior to the requirements phase. This will depend upon how well referencing and labeling has been provided in the earlier documents/ work products.
* **Forward traceability** this will depend upon how each requirement in the SRS is labeled/number. Forward traceability is extremely important as this is one of the ways of tracing a requirement to its implementation.
  + 1. **Feasibility**

Though it may not be possible to confirm the feasibility of implementation of all requirements, any requirement which is apparently infeasible, should be eliminated from the SRS, after giving the reasoning.

* + 1. **Testability**

The SRS needs to be stated in such a way that **it is possible for the tester to create test plan and confirm whether the requirements are met or not.** Typical statements in a SRS which may be non testable or statements which contain phrases like “**user friendly”, “reasonable response”, “fairly accurate”.** Statements that are non testable and non verifiable need to be identified and removed or revised.

* 1. **Contents Of SRS Document**

The SRS template should be selected or defined at the start of the requirements elicitation activities, so that the contents of the documents evolve and buildup as the requirements elicitation and analysis activities progress. Following are the contents of the SRS:

**Introduction**

The introduction sets the tone of the document by providing a background to the customer, users, development team and the context of the proposed system. This also describes the purpose of the SRS, the intended audience, the structure of the SRS, the terminology and abbreviations used, any conventions used and lists other documents referenced.

* **Purpose of this document**  
  Describes the purpose of the document, and the intended audience.
* **Scope**   
  Describes the scope of this requirements definition effort. Introduces the requirements elicitation team, including users, customers, system engineers, and developers.

This section also details **any constraints** that were placed upon the requirements elicitation process, such as schedules, costs, or the software engineering environment used to develop requirements.

* **Overview**  
  Provides a brief overview of the product defined as a result of the requirements elicitation process.
* **Business-Context**  
  Provides an overview of the **business organization sponsoring the development of this product**. This overview should include the **business's mission statement** and its organizational objectives or goals.

**System Description**

In this the **current system and its problems are described**. The goals of the proposed system are described in business terms along with an overview of the proposed system. The characteristics of the user are documented in terms of familiarity with computerize system and automation and finally all assumptions on which the SRS depends are listed.

* **Product-Functions**  
  Describes the general functionality of the product.
* **Similar-System-Information**  
  Describes the relationship of this product with any other products. Specifies if this product is intended to be stand-alone, or else used as a component of a larger product. If the latter, this section discusses the relationship of this product to the larger product.
* **User-Characteristics**  
  Describes the features of the user community, including their expected expertise with software systems and the application domain.
* **User-Problem-Statement**  
  This section describes the essential problem(s) currently confronted by the user community.
* **User-Objectives**  
  This section describes the set of objectives and requirements for the system from the user's perspective. It may include a "wish list" of desirable characteristics, along with more feasible solutions that are in line with the business objectives.
* **General-Constraints**  
  Lists general constraints placed upon the design team, including speed requirements, industry protocols, hardware platforms, and so forth.

**Functional Requirements**

This starts with a description of how the system is partitioned into modules (or subsystems). Further more each module is described in terms of detailed requirements that the module needs to satisfy. The functional requirements may be explained with the help of diagrams, use cases, schematics, charts and layouts. The structure of this content would also depend on the modeling technique used during the requirements analysis stage. How each module satisfies the stated goals is also documented at the end of the functional description of each module. Small system may not be partitioned into modules.

This section lists the functional requirements in ranked order. Functional requirements describes the possible effects of a software system, in other words, what the system must accomplish. Other kinds of requirements (such as interface requirements, performance requirements, or reliability requirements) describe how the system accomplishes its functional requirements.

**Non-Functional Requirements**

Non-functional requirements are characteristics of system to be achieved that are not related to the functionality. Non-functional requirements define the system properties and constraints and also contain system “Quality” attributes. Non-functional requirements should be stated separately from the functional requirements so that the users can confirm during the requirements verification stage, the designers can focus on achieving these requirements and testers can specifically conduct focused tests to confirm the achievement of the non functional requirements.

This content of the SRS document further consist of the sub-contents which are described below:

**Performance** Performance or speed requirements are typically expressed as “processed transactions per second” or “response time from the system for a user event” or “screen refresh time” or a combination of these.

**Resource Utilization Efficiency** this is a measure of how efficiently the system utilizes the CPU, the RAM, and communications bandwidth and disk space.

**Security** security is the ability of the system to protect its functions and data from access, damage and misuse.

**Safety** Safety requirements are those that are concerned with possible loss, damage, or harm that could result from the use of the system.

**Capacity** This is the volume of transactions and the volume of data that the system will support, giving due consideration to future increase/decrease in the volume.

**Interfaces** The other known and unknown systems that our system must interface with and the types of interface that shall operate.

**Availability** Availability is the requirement for the system to be functional and operating during a specified period.

**Reliability** Reliability is the characteristic of a system that increase its ability to execute without failure for a longer duration. A system’s reliability can be measured in one or more of the following ways:

* Mean time to failure (MTTF). MTTF is the average time that a system operates before encountering a failure. Reliable systems have a large MTTF.
* Average defects reported in a time period.

**Accuracy** The accuracy of the information stored and processing done needs to be specified. This can be done in terms of allowed “margin of error”.

**Reusability** Many software projects have an additional goal of building a system whose component can easily be modified and ported to other systems with no or minimal modifications. This increases the discipline required in creating modular, reusable software much beyond the need of the system being developed.

**Ease of Use** An easy to use system results in (1) reduced training time (2) reduced time to perform user operations. Some of the features put in to satisfy the first goal might result in deterioration of the second goal.

**Interoperability** Interoperability is the characteristics of the system that allows it to exchange data or services with other systems.

**Portability** System designed on one plate form will need to be ported to other plate forms due to changes in technology or need for handling larger volumes or increasing features and functionality.

**Privacy** This is closely related to the security requirements. In many countries data acquire through one channel cannot be used for other purposes without explicit permission. In such cases privacy requirements need to be stated as non functional requirements.

**Expandability** Expandability is the degree to which architectural, data or procedural design can be extended.

**Maintainability** Higher maintainability increases the speed with which the developers/maintainers make an enhancement or fix a defect in the system. High maintainability is one of the ways to increase the availability of the system and also essential for system that are likely to undergo frequent modification.

**Testability** Increased testability ensures that new modules can be integrated and the whole system can be tested with the ease. To improve testability, designers need to increase the modularity, programmers need to reduce the cohesion and testers need to create the test work products so that focused testing can be performed on specific areas of the system.

**Interface Requirements**

This section describes how the software interfaces with other software products or users for input or output. Examples of such interfaces include library routines, token streams, shared memory, data streams, and so forth.

**User Interfaces**  
Describes how this product interfaces with the user.

* **GUI**  
  Describes the graphical user interface if present.
* **CLI**  
  Describes the command-line interface if present.
* **API**  
  Describes the application programming interface, if present.

**Hardware Interfaces**  
Describes interfaces to hardware devices.

**Communications Interfaces**  
Describes network interfaces.

**Software Interfaces**  
Describes any remaining software interfaces not included above.

**Design Constraints**

In this content of SRS should provide a description of the issues that are likely to limit or act as a constraint on the designers. These could include:

* **International, National and local Laws.**
* **Hardware limitations**
* **Interfaces to other systems.**
* **Audit requirements.**

In some documentation the design constraints are included in the addition to the non-functional requirements.

**Project Requirements**

This content is treated as optional it may or may not be include in SRS depends upon the way of writing the SRS means on which it is focusing. So here, many of the requirements for this project execution may be known at this point in time. If known and not present in any document, they can be (optionally) specified in SRS. The project requirements at this stage typically include the size, effort and cost estimates, high-level schedules and the development plate form on which the system is to be developed. The acceptance criteria can also be specified in this section.

**Preliminary Schedule**

This section provides an initial version of the project plan, including the major tasks to be accomplished, their interdependencies, and their tentative start/stop dates. The plan also includes information on hardware, software, and wetware resource requirements.

The project plan should be accompanied by one or more PERT or GANTT charts.

**Preliminary Budget**

This section provides an initial budget for the project, itemized by cost factor.

**Appendices**

Specify other useful information for understanding the requirements. All SRS documents should include at least the following two appendices:

* **Definitions, Acronyms, Abbreviations**  
  Provides definitions of unfamiliar definitions, terms, and acronyms.
* **References**  
  Provides complete citations to all documents and meetings referenced or used in the preparation of this document.

**4.4 Importance Of Requirements Documentation**

The documentation is important in every field, this is not only essential for the software development process if you are considering the medical side the doctors are use to keep the record of their patients this is the documentation for the patient’s record, the manuals which are provide by the company with each product that they sale this is the documentation of that product, means the documentation describes the space for which it is build, for what it will be used and how it will be used this is said to be the documentation. So here in the software development process the documentation acts very much important role, in simple words we can say that the documentation describes the scenario of the purpose of the whole system which will be build, the document of any system highlights the requirements of the customer that what the customer actually wants?, and it also shows the whole way of the implementation of the system means how the system will be implemented what it will contain how it will work, the way how the end user will operate it means just like a manual is said to be the documentation. As we mentioned earlier that the documentation is necessary in every field either it is related to software or any other department such as medical etc. The physicians are use to make the document of their patients and so through that record they can easily identify the previous reports means that is said to be reusability in terms of software engineering, that thing also saves the time and cost to review and manage the system, in those cases specially when the requirements of the customers are going to be change day by day according to the market means the requirements are not freeze.

A significant portion of most software development efforts is devoted to documentation. In addition to improving program quality, good documentation improves software usability and maintainability; these are very important for reducing long-term software related expenses. Documentation can play a major positive role in the software development process. Unfortunately, very often, the documentation is one of the final steps in software development. If the documentation is written by the developers when the project is nearing completion, the document is often useful to people who know the software well, but usually hard for others to understand. Because of the complexity of mathematical models that are usually used in engineering and scientific simulations, any way to make requirements documentation easier to understand and use during software development could dramatically reduce the number of errors found in such software.

If the documentation is well written with a particular format then that is very much use full for further phases or stages which are incoming after that, such as the developer can easily review the system, the users can understand what they actually expected from the system, the analyst can easily know what the user’s need that have to be build etc. so after that the complexity to develop the system will reduce.

* 1. **Summary**

Documentation is the descriptive information (e-g hardcopy manuals, online help files, web sites) that portrays the use and/or operation of the system. documentation describes the space for which it is build, for what it will be used and how the particular system will be used by the users. The requirements document usually called the software requirements specification (SRS). The software requirements specification (SRS) is produced at the end of the elicitation and analysis tasks. Documentation of any system describes the purpose and implementation requirements of the system. Further the contents are the major part of the documentation means the document is fully depend on what the document actually contains.

A “good” SRS document has the following characteristics: (1) Completeness (2) Clarity (3) Correctness (4) Consistency (5) Modifiability (6)Traceability (7) Feasibility (8) Testability.

The SRS template should be selected or defined at the start of the requirements elicitation activities, so that the contents of the documents evolve and buildup as the requirements elicitation and analysis activities progress. A significant portion of most software development efforts is devoted to documentation. In addition to improving program quality, good documentation improves software usability and maintainability.