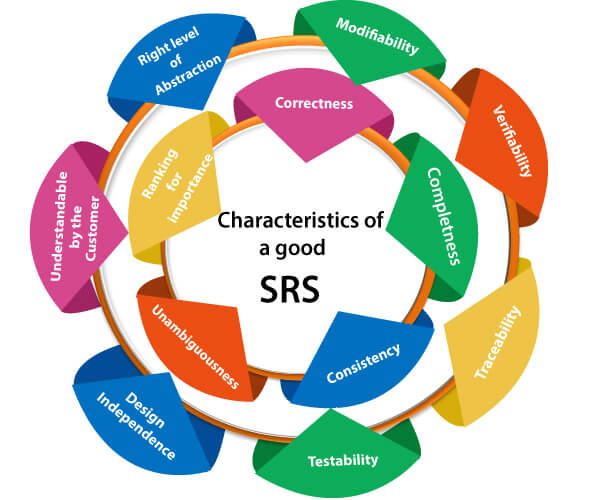
Software Requirement Specifications

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analysed. **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.Characteristics of good SRS



**Following are the features of a good SRS document:**

**1. Correctness:** User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

**2. Completeness:** The SRS is complete if, and only if, it includes the following elements:

**(1).** All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

**(2).** Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

#### Note: It is essential to specify the responses to both valid and invalid values.

**(3).** Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

**3. Consistency:** The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

**(1).** The specified characteristics of real-world objects may conflicts. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

**(2).** There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

**(3).** Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

**4. Unambiguousness:** SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

**5. Ranking for importance and stability:** The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

**6. Modifiability:** SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

**7. Verifiability:** SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

**8. Traceability:** The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

**There are two types of Traceability:**

**1. Backward Traceability:** This depends upon each requirement explicitly referencing its source in earlier documents.

**2. Forward Traceability:** This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

**9. Design Independence:** There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

**10. Testability:** An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

**11. Understandable by the customer:** An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

**12. The right level of abstraction:** If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas,for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

## Properties of a good SRS document

**The essential properties of a good SRS document are the following:**

**Concise:** The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

**Structured:** It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

**Black-box view:** It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behaviour of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behaviour of the system. For this reason, the SRS report is also known as the black-box specification of a system.

**Conceptual integrity:** It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

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

## SRS: Software Requirements Specification

Software Requirements Specification (SRS) is a comprehensive document that outlines the requirements and expectations for a software project. It is a crucial step in the software development process as it sets the foundation for the entire project and ensures that everyone involved understands the end product. This article will look closely at SRS, its benefits, and what should be included in a comprehensive SRS document.

## What is SRS?

It is a comprehensive document that covers all aspects of the software project, from the project's goals and objectives to the software's specific requirements. It outlines the functional and non-functional requirements of the software and serves as a blueprint for the entire project. The SRS should be written in a way that is clear, concise, and easy to understand so that everyone involved in the project has a clear understanding of what is expected.

## What Should Be Included in a Comprehensive SRS Document?

A comprehensive SRS document should include the following elements:

**Introduction**

This section should provide an overview of the software project, including its goals and objectives.

**Requirements**

This section should outline the functional and non-functional requirements of the software. This may include a user interface description, performance requirements, and other functional requirements.

**Use Cases**

This section should describe the different scenarios in which the software will be used, including user interactions and system behaviours.

**Design Constraints**

This section should outline any design constraints or limitations that must be considered when developing the software.

**User Requirements**

This section should describe the requirements for the software from the user's perspective, including usability, accessibility, and security requirements.

**System Requirements**

This section should describe the requirements for the software from a system perspective, including performance, scalability, and reliability requirements.

**Technical Requirements**

This section outlines any technical requirements that the software must meet. It should include information on software architecture, design, and implementation.

**Quality Requirements**

This section outlines the quality requirements that the software must meet. It should include information on reliability, availability, maintainability, and security.

**Acceptance Criteria**

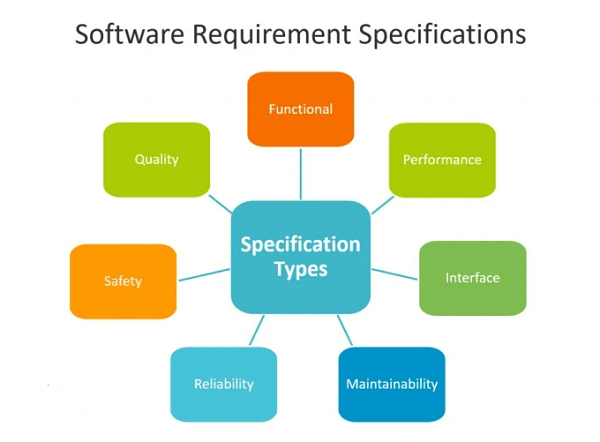
This section should outline the criteria that must be met for the software to be accepted by the customer or end user.

**Glossary**

This section should define any technical terms used in the SRS document.

**Project Schedule**

This section outlines the project schedule for the software development process. It should include project milestones, deadlines, and a timeline for completion. It is making any changes to the project requirements, scope, or timeline.



### Conclusion

Software Requirements Specification is a crucial step in the software development process. It provides a clear understanding of the goals and objectives of the project and ensures that everyone involved has a clear understanding of what the end product should be. A comprehensive SRS document should include an introduction and requirements.

## Requirements Smell in Software Requirements Specification

However, just like any other document, it can have issues and challenges that can negatively impact the quality of the final software product. One of these issues is requirements smells, which are signs that something is wrong with the requirements in the SRS. This article will look at some of the standard requirements for smells and how they can be addressed.

**Ambiguity**

Ambiguous requirements can lead to misunderstandings, miscommunication, and, ultimately, a failed project. When a requirement is ambiguous, it needs to be clarified what it means and what it requires. This can result in different interpretations by stakeholders, leading to confusion and rework. To avoid ambiguity, requirements should be clear, concise, and specific.

**Vaguenes**

Vague requirements are those that lack detail and precision. They may need to specify what the software should do or how it should do it. Vague requirements can lead to misunderstandings and misinterpretations, causing delays and rework. To avoid vagueness, requirements should be clear and specific, with sufficient detail to allow for implementation.

**Inconsistency**

Inconsistent requirements can arise when stakeholders have different views on what is required. This can lead to misunderstandings and misinterpretations, resulting in rework and delays. To avoid inconsistency, requirements should be consistent and coherent, clearly understanding what is required and how it should be implemented.

**Unclear Assumptions**

Requirements can contain assumptions that need to be explicitly stated. This can lead to misunderstandings and misinterpretations. By using an SRS document, organizations can ensure that the software being developed is of high quality, meets the needs of the stakeholders, and is developed efficiently.

**Increased Transparency**

An SRS document provides a clear understanding of the project requirements, providing increased transparency for stakeholders. This increased transparency can build trust and confidence in the development team and the project.

**Improved Decision-Making**

An SRS document can provide stakeholders with the information they need to make informed decisions.

## Project Requirements and Scope

Stakeholders can make better-informed decisions by clearly understanding the project requirements and scope.

**Better Project Outcomes**

By clearly understanding the project requirements, the SRS document can ensure that the developed software meets the stakeholders' expectations and provides a good user experience.

**Reduced Risks**

An SRS document can help to reduce risks associated with the software development process. By clearly understanding the project requirements, the SRS document can reduce the chances of misunderstandings, rework, and delays.

## Creating and maintaining the SRS document

Clearly understand Creating and maintaining the SRS document is a collaborative effort involving multiple parties, including the development team, stakeholders, and subject matter experts. Here are some best practices for creating and maintaining an SRS document:

**Involving Stakeholders**

Involve stakeholders in the SRS document creation process to ensure that the software requirements accurately reflect their needs and expectations.

**Use a Template**

Use a well-defined template for creating the SRS document. This helps to ensure that all relevant information is captured and that the document is organized in a consistent and easily understandable format.

**Define the Requirements Clearly**

The requirements should be defined clearly and precisely, avoiding ambiguity and ensuring that they are measurable and verifiable.

**Prioritize Requirements**

Prioritize requirements based on their importance and feasibility.

Regularly review and update the SRS document to ensure that it remains relevant and accurate. This helps keep all parties informed of any project requirements or scope changes.

**Get approval from stakeholders**

Obtain approval from stakeholders for the final version of the SRS document to ensure that everyone is aligned on the project requirements.

**Use Clear and Concise Language**

Use clear and concise language in the SRS document to ensure that it is easily understood by all parties involved in the software development process. By following these best practices, organizations can ensure that the SRS document is comprehensive, accurate, and up-to-date, serving as an essential reference throughout the software development process.

## Steps To Write an SRS Document

An SRS (Software Requirements Specification) document describes a specific project's software requirements and specifications. Here are the steps to write an SRS document:

1. **Define the purpose and scope of the software:** What problem does the software aim to solve, and what are the boundaries of its functionality?
2. **Identify the stakeholders:** Who will be using the software, and what are their requirements?
3. **Gather requirements:** Conduct interviews, surveys, and focus groups to gather information about the software requirements.
4. **Organize the requirements:** Categorize them into functional and non-functional requirements and prioritize them based on their importance.
5. **Write the document:** Use a clear and concise writing style to describe each requirement, including the purpose, inputs, outputs, and constraints.
6. **Review and validate:** Have stakeholders review the document to ensure it meets their requirements and is accurate and complete.
7. **Update the document:** Continuously update the document as the project progresses, and new requirements are identified.

Keeping the SRS document updated throughout development is essential to ensure the final product meets the desired specifications and requirements. Here are a few additional details that may be useful when writing an SRS document:

1. **Use a standard format:** Use a standard format to make the document easy to read and understand. A standard format for SRS documents includes the following sections: Introduction, Overall Description, Specific Requirements, Assumptions and Dependencies, and Appendices.
2. **Be specific and clear:** Be specific and transparent in your requirement description. Use precise language and avoid ambiguity. Specify any constraints, such as performance, usability, and security requirements.
3. **Use diagrams and illustrations:** Use diagrams and illustrations, such as flow charts and wireframes, to help explain complex requirements and relationships between different software parts.
4. **Include non-functional requirements:** Non-functional requirements, such as performance, security, and usability, are just as important as functional requirements and should be included in the SRS document.
5. **Consider user requirements:** Consider the requirements of the end-users when writing the SRS document. These requirements will drive the software's functionality and should be given proper attention.
6. **Please focus on the big picture:** While it's essential to include specific details, the SRS document should also focus on the big picture and provide an overall understanding of the software requirements and how they fit into the larger project.
7. **Get stakeholder approval:** Once the SRS document is complete, get approval from the stakeholders to ensure that all requirements have been captured and the document accurately reflects the desired outcome.

By following these guidelines, you can create a comprehensive and practical SRS document that will serve as a valuable resource throughout the software development process. An SRS document can be written in either Microsoft Word or specialized requirement management software.

## Benefits of SRS

The benefits of having a well-written SRS are numerous. Here are some key benefits: Clear understanding of project goals and objectives: The SRS outlines the goals and objectives of the project, ensuring that everyone involved clearly understands what is expected. Improved communication: A well-written SRS serves as a common reference point for all stakeholders, improving communication and reducing misunderstandings.

**Reduced Development Time**

The SRS provides a clear roadmap for the development process, reducing development time and improving efficiency.

**Better Project Planning**

The SRS provides a clear understanding of the project's scope, allowing for better project planning and resource allocation.

**Improved Quality**

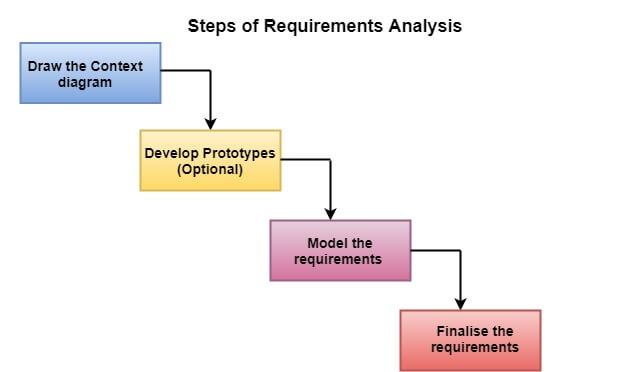
A well-written SRS provides a clear understanding of the requirements, reducing the risk of misunderstandings and improving the quality of the end product.

**Better Customer Satisfaction**

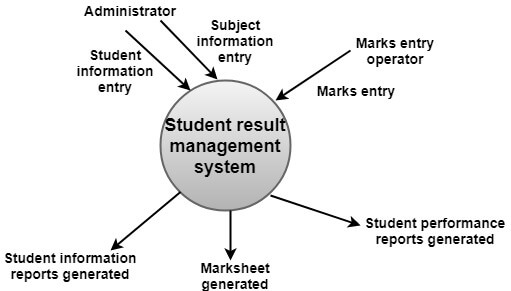
A well-written SRS ensures that the software meets the customer's expectations, leading to better customer satisfaction.

# Requirements Analysis

Requirement analysis is significant and essential activity after elicitation. We analyse, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

**The various steps of requirement analysis are shown in fig:**

**(i) Draw the context diagram:** The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below:



**(ii) Development of a Prototype (optional):** One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though a person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

**(iii) Model the requirements:** This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

**(iv) Finalise the requirements:** After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.