



**Experiment No-4**

**Title: Preparation of Software Requirements  
Specifications (SRS).**



---

**Aim: Preparation of Software Requirements Specifications (SRS).**

---

**Resources needed: LaTeX Editor, Internet Browser**

---

## Theory

Requirements specify how the target system should behave. It specifies what to do, but not how to do it. Requirements engineering refers to the process of understanding what a customer expects from the system to be developed, and to document them in a standard and easily readable and understandable format. This documentation will serve as reference for the subsequent design, implementation and verification of the system.

It is necessary and important that before we start planning, design and implementation of the software system for our client, we are clear about its requirements. If we don't have a clear vision of what is to be developed and what all features are expected, there would be serious problems and customer dissatisfaction as well.

## Characteristics of Requirements

Requirements gathered for any new system to be developed should exhibit the following three properties:

**1. Unambiguity:** There should not be any ambiguity about what a system to be developed should do. For example, consider you are developing a web application for your client. The client requires that enough people should be able to access the application simultaneously. What's the "enough number of people"? That could mean 10 to you but, perhaps, 100 to the client. There's an ambiguity.

**2. Consistency:** To illustrate this, consider the automation of a nuclear plant. Suppose one of the clients says that if the radiation level inside the plant exceeds R1, all reactors should be shut down. However, another person from the client side suggests that the threshold radiation level should be R2. Thus, there is an inconsistency between the two end users regarding what they consider as threshold level of radiation.

**3. Completeness:** A particular requirement for a system should specify what the system should do and also what it should not. For example, consider software to be developed for ATMs. If a customer enters an amount greater than the maximum permissible withdrawal amount, the ATM should display an error message, and it should not dispense any cash.

## **Categorization of Requirements**

Based on the target audience or subject matter, requirements can be classified into different types, as stated below:

- 1. User requirements:** They are written in natural language so that both customers can verify their requirements have been correctly identified
- 2. System requirements:** They are written involving technical terms and/or specifications, and are meant for the development or testing teams

Requirements can be classified into two groups based on what they describe:

- 1. Functional requirements (FRs):** These describe the functionality of a system -- how a system should react to a particular set of inputs and what should be the corresponding output.
- 2. Non-functional requirements (NFRs):** They are not directly related to what functionalities are expected from the system. However, NFRs could typically define how the system should behave under certain situations. For example, a NFR could say that the system should work with 128MB RAM. Under such condition, a NFR could be more critical than a FR.

## **IEEE SRS document (Relevant IEEE standards: IEEE-830[5])**

The SRS is a specification of the requirements for the software “product” you will produce in your project. The basic issues to be addressed are:

- a) Functionality. What is the software supposed to do?
- b) External interfaces. How should the software interact with people, the operating system, hardware, networks, and other software?
- c) Performance. What are the requirements for speed, availability, response time, recovery time of various software functions, etc.?
- d) Quality Attributes. What are the requirements for portability, correctness, maintainability, security, etc?
- e) Design constraints imposed on an implementation. Is there a requirement for a particular programming language? Are there resource limits (such as disk or memory size)? Must it run on a particular operating system? Must it inter-operate with particular web browsers? etc.

The SRS contains requirements, and not your design solutions; it is the “what” and not the “how” of your project. The information is collected from the project client. In a good SRS, the requirements should be: Correct, Unambiguous, Complete, Consistent, and Ranked for importance, Verifiable, and Traceable.

## 1 INTRODUCTION

### 1.1 Product Overview

Clearly define the purpose of the software, the environment it will run in, and who will be using it (in terms of their educational level, experience, and technical expertise). Do not go into detail, but outline the general requirements in a way that provides the reasons why specific requirements are later specified in the SRS.

## 2 SPECIFIC REQUIREMENTS

This section of the SRS should contain all of the software requirements to a level of detail sufficient to:

1. enable the SRS to be checked by the originator of the original system requirements (usually the project supervisor),
2. enable designers to design a system to satisfy those requirements, and
3. enable testers to test that the system satisfies those requirements.

Throughout this section, every stated requirement should be externally perceivable by users, or other external systems. This ensures that all features are testable.

### 2.1 External Interface Requirements

#### 2.1.1 User Interfaces

This should specify the following:

a) The characteristics of the user interface. Include the characteristics necessary to accomplish the software requirements (for example: required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys).

b) All the aspects of optimizing the interface with the person who must use the system. This may simply comprise a list of do's and don'ts on how the system will appear to the user. Like all others, these requirements should be verifiable.

#### 2.1.2 Hardware Interfaces

This should specify the characteristics of each interface between the software and the hardware components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. Only use this section if your project requires specific hardware to be used – do not, for example, specify the hardware if there is a requirement to establish a network connection.

#### 2.1.3 Software Interfaces

This should specify the use of other required software which your software must interface with (for example: a database system, an operating system, or a mathematical package).

For each required software product, the following should be provided:

- Name and Version number.

For each interface, the following should be provided:

- Discussion of the purpose of the interfacing software as related to this software product.

- Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

#### 2.1.4 Communications Protocols

This should specify the various interfaces to communications such as local network protocols, etc. Make reference to any well-defined protocols, specifying exactly which parts or options of that protocol the software needs to support.

### 2.2 Software Product Features

This section should consist of a numbered list of required features.

Typically, all of the requirements that relate to a software product are not equally important. The importance of each feature should be indicated using one of the following terms: essential, important, or desirable.

Each feature requirement should include:

- ? ? a description of every input (stimulus) into the system,
- ? ? a description of every output (response) from the system,
- ? ? a description of every state change within the system,
- ? ? a description of all the functions performed by the system in response to an input or in support of an output.

**Functional requirements** should define the actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as “shall” statements starting with “The system shall”. These include

- a) Validity checks on the inputs
- b) Exact sequence of operations
- c) Responses to abnormal situations, including
  - 1) Overflow
  - 2) Communication facilities
  - 3) Error handling and recovery
- d) Effect of parameters
- e) Relationship of outputs to inputs, including
  - 1) Input/output sequences
  - 2) Formulas for input to output conversion

It may be appropriate to partition the functional requirements into sub functions or sub processes. This does not imply that the software design will also be partitioned that way.

### 2.3 Software System Attributes

#### 2.3.1 Reliability

Specify the required reliability of the final software system.

This is particular important for applications such as embedded software. It can be quantified using MTTF (Mean Time To Failure) measurements.

#### 2.3.2 Availability

Specify the required availability of the final software system: define requirements such as check pointing, recovery, and restart.

**2.3.3 Security**

Specify the factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure.

Specific requirements in this area could include the need to

- a) Utilize certain cryptographic techniques;
- b) Keep specific log or history data sets;
- c) Assign certain functions to different modules;
- d) Restrict communications between some areas of the program;
- e) Check data integrity for critical variables.

**2.3.4 Maintainability**

This should specify attributes of the software that relate to the ease of maintenance of the software itself. Specify any requirements for certain modularity, interfaces, complexity, etc that make the software easier to maintain.

Requirements should not be placed here just because they are thought to be good design practices.

**2.3.5 Portability**

This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include the following:

- a) Percentage of code that is host dependent;
- b) Use of a proven portable language;
- c) Use of a particular compiler or language subset;
- d) Use of a particular operating system.

**2.3.6 Performance**

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole.

Static numerical requirements may include, for example, the following: minimum number of simultaneous users, minimum data storage, etc.

Dynamic numerical requirements may include, for example, the required number of transactions per second for both normal and peak workload conditions, etc.

All of these requirements should be stated in measurable terms.

**2.4 Database Requirements**

This should specify the logical requirements for any information that is to be placed into a database.

This may include the following:

- a) Types of information used by various functions;
- b) Accessing capabilities;
- c) Data entities and their relationships;
- d) Integrity constraints.

### Activities:

Prepare SRS document for chosen problem definition in LaTeX.

---

**Results:** SRS Document in given format

### LaTeX Code:

```
\documentclass{article}
\usepackage{enumitem}

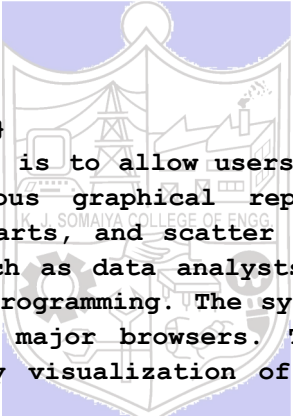
\title{Software Requirements Specification (SRS) for Data Visualization Tool}
\author{}
\date{}

\begin{document}

\maketitle

\section{Introduction}

\subsection{Product Overview}
The purpose of this software is to allow users to input numerical data and visualize it through various graphical representations, such as bar charts, line graphs, pie charts, and scatter plots. The tool is designed for non-technical users, such as data analysts or business professionals, with limited experience in programming. The system will run in a web-based environment, supporting all major browsers. The primary requirement for this tool is to enable easy visualization of data with minimal setup or training.
```



```
\section{Specific Requirements}

This section outlines the specific functional and non-functional requirements for the data visualization tool. The requirements are described to the level of detail necessary to allow verification, design, and testing.

\subsection{External Interface Requirements}

\subsubsection{User Interfaces}
\begin{itemize}
\item The user interface must provide a clear and simple dashboard where users can upload data files in CSV or Excel format.
\item The system shall display a list of available visualization types such as bar charts, pie charts, and line graphs in a sidebar.
\item Each visualization type shall have a preview of the graph and an option to customize axes, colors, and labels.
\item The system shall allow the user to zoom in and out of charts for better detail viewing.
```

\item The application shall include a "Save as Image" feature to export the chart in PNG, JPG, or SVG format.

\item Tooltips must provide context-sensitive help to guide users in selecting and customizing visualizations.

\item The system shall display error messages if the data is not compatible or contains invalid values.

\end{itemize}

#### \subsubsection{Hardware Interfaces}

The software must run on standard desktops or laptops with the following specifications:

\begin{itemize}

\item CPU: Minimum 1.5 GHz Processor

\item RAM: 4GB minimum

\item Storage: 50MB of free space

\item Display: Minimum resolution of 1024x768

\end{itemize}

The system does not require any specific hardware interface but must operate within the limitations of common modern computing environments.

#### \subsubsection{Software Interfaces}

\begin{itemize}

\item The software shall interface with web browsers, specifically Chrome, Firefox, Safari, and Edge.

\item The system shall be compatible with operating systems such as Windows, macOS, and Linux.

\item The software must be able to read data from CSV and Excel files.

\item The system shall utilize the D3.js JavaScript library for rendering the visualizations.

\item For storage and user data management, the software shall interface with a backend database like MySQL or MongoDB for saving user configurations and session data.

\end{itemize}

#### \subsubsection{Communications Protocols}

The software shall communicate over HTTPS to ensure secure transmission of data between the user's browser and the server. The system will follow RESTful APIs for fetching and submitting data to the backend.

#### \subsection{Software Product Features}

\begin{enumerate}

\item \textbf{Data Upload:} The system shall allow users to upload numerical data in CSV or Excel format.

\item \textbf{Visualization Types:} The system shall provide multiple chart types (bar charts, line graphs, pie charts, scatter plots).

\item \textbf{Chart Customization:} Users shall be able to modify the title, axes labels, colors, and chart types.

\item \textbf{Data Filtering:} Users shall be able to filter data by date ranges or specific values for better visualization.

\item \textbf{Export Feature:} Users shall be able to export visualized data as an image in PNG, JPG, or SVG format.



```
\item \textbf{Data Error Handling:} The system shall notify users if
the data format is incorrect or contains invalid entries.
\end{enumerate}
```

Each feature shall be implemented in accordance with the following rules:

```
\begin{itemize}
```

```
\item The system shall validate uploaded data to ensure it is numeric
and free of errors.
```

```
\item The system shall use predefined color schemes for different
types of visualizations (e.g., blue for bar charts, green for line
graphs).
```

```
\item The system shall provide real-time updates to the visualizations
as users modify input data.
```

```
\end{itemize}
```

```
\subsection{Software System Attributes}
```

```
\subsubsection{Reliability}
```

The software shall be reliable, with a target Mean Time To Failure (MTTF) of 1000 hours. It shall recover from crashes or unexpected shutdowns without loss of data.

```
\subsubsection{Availability}
```

The system shall be available 99.9\% of the time, with downtime limited to scheduled maintenance windows.

```
\subsubsection{Security}
```

```
\begin{itemize}
```

```
\item The software shall use HTTPS for secure communication between
the client and server.
```

```
\item The system shall require user authentication to access saved
data or settings.
```

```
\item All uploaded data shall be encrypted before being stored in the
database.
```

```
\item The system shall maintain an activity log for auditing purposes.
```

```
\end{itemize}
```

```
\subsubsection{Maintainability}
```

The software shall be modular, with clearly separated components for the frontend, backend, and data processing logic. It shall be easy to extend and maintain, with the code following best practices for readability and commenting.

```
\subsubsection{Portability}
```

The system shall be portable across all modern browsers (Chrome, Firefox, Safari, Edge) and operating systems (Windows, macOS, Linux).

```
\subsubsection{Performance}
```

```
\begin{itemize}
```

```
\item The system shall support up to 1000 concurrent users without
noticeable performance degradation.
```

```
\item The system shall render visualizations within 2 seconds after
data input or modification.
```

```
\end{itemize}
```

```
\subsection{Database Requirements}
```

The database shall store user profiles, preferences, and uploaded data. It shall support the following:

```
\begin{itemize}
```

```
  \item Storing of CSV and Excel data files.
```

```
  \item Support for querying numerical data based on user-defined filters.
```

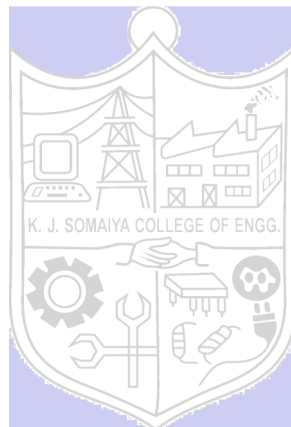
```
  \item Integrity constraints to ensure no invalid data is entered.
```

```
  \item Efficient indexing to allow fast retrieval of visualization data.
```

```
\end{itemize}
```

```
\end{document}
```

---



# Software Requirements Specification (SRS) for Data Visualization Tool

## 1 Introduction

### 1.1 Product Overview

The purpose of this software is to allow users to input numerical data and visualize it through various graphical representations, such as bar charts, line graphs, pie charts, and scatter plots. The tool is designed for non-technical users, such as data analysts or business professionals, with limited experience in programming. The system will run in a web-based environment, supporting all major browsers. The primary requirement for this tool is to enable easy visualization of data with minimal setup or training.

## 2 Specific Requirements

This section outlines the specific functional and non-functional requirements for the data visualization tool. The requirements are described to the level of detail necessary to allow verification, design, and testing.

### 2.1 External Interface Requirements

#### 2.1.1 User Interfaces

- The user interface must provide a clear and simple dashboard where users can upload data files in CSV or Excel format.
- The system shall display a list of available visualization types such as bar charts, pie charts, and line graphs in a sidebar.
- Each visualization type shall have a preview of the graph and an option to customize axes, colors, and labels.
- The system shall allow the user to zoom in and out of charts for better detail viewing.
- The application shall include a "Save as Image" feature to export the chart in PNG, JPG, or SVG format.

- Tooltips must provide context-sensitive help to guide users in selecting and customizing visualizations.
- The system shall display error messages if the data is not compatible or contains invalid values.

### **2.1.2 Hardware Interfaces**

The software must run on standard desktops or laptops with the following specifications:

- CPU: Minimum 1.5 GHz Processor
- RAM: 4GB minimum
- Storage: 50MB of free space
- Display: Minimum resolution of 1024x768

The system does not require any specific hardware interface but must operate within the limitations of common modern computing environments.

### **2.1.3 Software Interfaces**

- The software shall interface with web browsers, specifically Chrome, Firefox, Safari, and Edge.
- The system shall be compatible with operating systems such as Windows, macOS, and Linux.
- The software must be able to read data from CSV and Excel files.
- The system shall utilize the D3.js JavaScript library for rendering the visualizations.
- For storage and user data management, the software shall interface with a backend database like MySQL or MongoDB for saving user configurations and session data.

### **2.1.4 Communications Protocols**

The software shall communicate over HTTPS to ensure secure transmission of data between the user's browser and the server. The system will follow RESTful APIs for fetching and submitting data to the backend.

## 2.2 Software Product Features

1. **Data Upload:** The system shall allow users to upload numerical data in CSV or Excel format.
2. **Visualization Types:** The system shall provide multiple chart types (bar charts, line graphs, pie charts, scatter plots).
3. **Chart Customization:** Users shall be able to modify the title, axes labels, colors, and chart types.
4. **Data Filtering:** Users shall be able to filter data by date ranges or specific values for better visualization.
5. **Export Feature:** Users shall be able to export visualized data as an image in PNG, JPG, or SVG format.
6. **Data Error Handling:** The system shall notify users if the data format is incorrect or contains invalid entries.

Each feature shall be implemented in accordance with the following rules:

- The system shall validate uploaded data to ensure it is numeric and free of errors.
- The system shall use predefined color schemes for different types of visualizations (e.g., blue for bar charts, green for line graphs).
- The system shall provide real-time updates to the visualizations as users modify input data.

## 2.3 Software System Attributes

### 2.3.1 Reliability

The software shall be reliable, with a target Mean Time To Failure (MTTF) of 1000 hours. It shall recover from crashes or unexpected shutdowns without loss of data.

### 2.3.2 Availability

The system shall be available 99.9% of the time, with downtime limited to scheduled maintenance windows.

### 2.3.3 Security

- The software shall use HTTPS for secure communication between the client and server.
- The system shall require user authentication to access saved data or settings.

- All uploaded data shall be encrypted before being stored in the database.
- The system shall maintain an activity log for auditing purposes.

#### **2.3.4 Maintainability**

The software shall be modular, with clearly separated components for the front-end, backend, and data processing logic. It shall be easy to extend and maintain, with the code following best practices for readability and commenting.

#### **2.3.5 Portability**

The system shall be portable across all modern browsers (Chrome, Firefox, Safari, Edge) and operating systems (Windows, macOS, Linux).

#### **2.3.6 Performance**

- The system shall support up to 1000 concurrent users without noticeable performance degradation.
- The system shall render visualizations within 2 seconds after data input or modification.

### **2.4 Database Requirements**

The database shall store user profiles, preferences, and uploaded data. It shall support the following:

- Storing of CSV and Excel data files.
- Support for querying numerical data based on user-defined filters.
- Integrity constraints to ensure no invalid data is entered.
- Efficient indexing to allow fast retrieval of visualization data.

**Questions:****1. Explain various steps involved in Requirement Engineering.**

**Ans:** Requirement Engineering is a systematic process of gathering, analyzing, documenting, and managing the requirements of a software system. The steps involved in Requirement Engineering are as follows:

- 1) **Requirement Elicitation:** This is the first step where the requirements are gathered from stakeholders, including clients, users, and domain experts. Techniques like interviews, questionnaires, brainstorming, and observation are used to collect information.
- 2) **Requirement Analysis:** The gathered requirements are analyzed to identify inconsistencies, ambiguities, and conflicts. The goal is to ensure that the requirements are clear, consistent, and feasible.
- 3) **Requirement Specification:** The analyzed requirements are documented in a structured format, such as a Software Requirements Specification (SRS) document. This document serves as a reference for developers, testers, and other stakeholders.
- 4) **Requirement Validation:** The documented requirements are reviewed and validated to ensure they meet the stakeholders' needs. Techniques like prototyping, reviews, and walkthroughs are used to validate the requirements.
- 5) **Requirement Management:** This involves tracking and managing changes to the requirements throughout the software development lifecycle. Tools like requirement traceability matrices are used to ensure that all requirements are addressed.

**Outcomes: CO3 — Demonstrate requirements, modeling and design of a system.**

**Conclusion: (Conclusion to be based on the outcomes achieved)**

The preparation of the Software Requirements Specification (SRS) document is a critical step in the software development process. It ensures that all stakeholders have a clear and unambiguous understanding of the system's requirements. By following the IEEE 830 standard, we were able to create a comprehensive SRS document for the "Text to VR: Visualizing Numerical Data" project. The document outlines the functional and nonfunctional requirements, external interfaces, and system attributes, ensuring that the development team has a clear roadmap for implementation. The use of LaTeX for documentation ensured a professional and standardized format. This exercise has reinforced the importance of thorough requirement engineering in delivering a successful software product.

**Grade: AA / AB / BB / BC / CC / CD / DD**

**Signature of faculty in-charge with date**

---

**References:**

**Books:**

1. Roger S. Pressman, Software Engineering: A practitioners Approach, 7th Edition, McGraw Hill, 2010.
  2. Technical report on Guidelines for Documents Produced by Student Projects In Software Engineering based on IEEE standards
  3. <https://www.sharelatex.com/>
- 

