Controls Analysis

Unni Kurumbail

**Software Requirements Specification**

**Document**

**Version: (0.0)** **Date: (Jun 27 2019)**

**Table of Contents**

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, and Abbreviations

1.4 References

1.5 Overview

2. The Overall Description

2.1 Product Perspective

2.1.1 System Interfaces

2.1.2 Interfaces

2.1.3 Hardware Interfaces

2.1.4 Software Interfaces

2.1.5 Communications Interfaces

2.1.6 Memory Constraints

2.1.7 Operations

2.1.8 Site Adaptation Requirements

2.2 Product Functions

2.3 User Characteristics

2.4 Constraints

2.5 Assumptions and Dependencies

2.6 Apportioning of Requirements

3. Specific Requirements

3.1 External interfaces

3.2 Functions

3.3 Performance Requirements

3.4 Logical Database Requirements

3.5 Design Constraints

3.5.1 Standards Compliance

3.6 Software System Attributes

3.6.1 Reliability

3.6.2 Availability

3.6.3 Security

3.6.4 Maintainability

3.6.5 Portability

3.7 Organizing the Specific Requirements

3.7.1 System Mode

3.7.2 User Class

3.7.3 Objects

3.7.4 Feature

3.7.5 Stimulus

3.7.6 Response

3.7.7 Functional Hierarchy

3.8 Additional Comments

4. Change Management Process

5. Document Approvals

6. Supporting Information

# 1. Introduction

*The following subsections of the Software Requirements Specifications (SRS) document should provide an overview of the entire SRS. The thing to keep in mind as you write this document is that you are telling what the system must do – so that designers can ultimately build it. Do not use this document for design!!!*

## 1.1 Purpose

The purpose of this SRS is to explain in detail the expected functionalities of the Controls Analysis software application being developed. It will list the expected model inputs and the expected UI. The intended audience is the developer of the software.

## 1.2 Scope

## 

The controls analysis software will take in model functions (f(t)) and disturbance requirements from users in a pre-defined format and will produce graphs of the Process Variable outputs as a function of time based on the control strategy of choice. The goal is to develop a robust and speedy software that can accurately model how the system will react to each implementable controls strategy. In doing so, the user will be able to make a better decision about what sort of controls system will be useful in their process, and whether any advantage will be conferred on them by using a more advanced control model.

## 1.3 Definitions, Acronyms, and Abbreviations.

*Provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendices in the SRS or by reference to documents. This information may be provided by reference to an Appendix.*

## 1.4 References

*In this subsection:*

*(1) Provide a complete list of all documents referenced elsewhere in the SRS*

*(2) Identify each document by title, report number (if applicable), date, and publishing organization*

1. *Specify the sources from which the references can be obtained.*

*This information can be provided by reference to an appendix or to another document. If your application uses specific protocols or RFC’s, then reference them here so designers know where to find them.*

## 1.5 Overview

*In this subsection:*

1. *Describe what the rest of the SRS contains*
2. *Explain how the SRS is organized*

*Don’t rehash the table of contents here. Point people to the parts of the document they are most concerned with. Customers/potential users care about section 2, developers care about section 3.*

# 2. The Overall Description

*The major requirements of the project are:*

* *The ability to handle routine process models*
* *The ability to utilize a few standard controls models*
* *The ability to offer modifiable parameters to adjust for unique user needs*
* *An intuitive and appealing GUI that the user can view multiple graphs on*
* *Interfaces that enable the implementation of complex control algorithms at a future time*
* *The ability to run on a Web Browser*

## 2.1 Product Perspective

The product will be a stand-alone software application that only serves these specific needs. It will consist of two major pieces of development:

1. A graphical interface that runs on a Web Browser
2. A back-end that performs the model manipulations, error handling, and control algorithm applications

Backend

GUI (Web Browser)

Model

Error handling, Internal calculations

Control Parameters

### 2.1.1 System Interfaces

* External Interface: Google Chrome Web Browser

### 

### 2.1.2 Interfaces

* GUI that contains graphs and modifiable parameters

### 2.1.3 Hardware Interfaces

The system does not have any known hardware interfaces.

### 

### 2.1.4 Software Interfaces

1. Google Chrome v75.0.3770.100 for Windows. The communication will take place via GET requests from the User (operating through buttons on the screen).

### 2.1.5 Communications Interfaces

*The Users will interact with the program via a Web Browser, accessed via the internet.*

### 2.1.6 Memory Constraints

*There are no known memory constraints*

### 2.1.7 Operations

*Specify the normal and special operations required by the user such as:*

1. *The various modes of operations in the user organization*
2. *Periods of interactive operations and periods of unattended operations*
3. *Data processing support functions*
4. *Backup and recovery operations*

*(Note: This is sometimes specified as part of the User Interfaces section.) If you separate this from the UI stuff earlier, then cover business process type stuff that would impact the design. For instance, if the company brings all their systems down at midnight for data backup that might impact the design. These are all the work tasks that impact the design of an application, but which might not be located in software.*

### 2.1.8 Site Adaptation Requirements

*In this section:*

1. *Define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode*
2. *Specify the site or mission-related features that should be modified to adapt the software to a particular installation*

*If any modifications to the customer’s work area would be required by your system, then document that here. For instance, “A 100Kw backup generator and 10000 BTU air conditioning system must be installed at the user site prior to software installation”.*

*This could also be software-specific like, “New data tables created for this system must be installed on the company’s existing DB server and populated prior to system activation.” Any equipment the customer would need to buy or any software setup that needs to be done so that your system will install and operate correctly should be documented here.*

## 2.2 Product Functions

*Provide a summary of the major functions that the software will perform. Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product.*

*For clarity:*

1. *The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.*
2. *Textual or graphic methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product but simply shows the logical relationships among variables.*

*AH, Finally the real meat of section 2. This describes the functionality of the system in the language of the customer. What specifically does the system that will be designed have to do? Drawings are good, but remember this is a description of what the system needs to do, not how you are going to build it. (That comes in the design document).*

## 2.3 User Characteristics

*Describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. Do not state specific requirements but rather provide the reasons why certain specific requirements are later specified in section 3.*

*What is it about your potential user base that will impact the design? Their experience and comfort with technology will drive UI design. Other characteristics might actually influence internal design of the system.*

## 2.4 Constraints

*Provide a general description of any other items that will limit the developer's options. These can include:*

*(1) Regulatory policies*

*(2) Hardware limitations (for example, signal timing requirements)*

*(3) Interface to other applications*

*(4) Parallel operation*

*(5) Audit functions*

*(6) Control functions*

*(7) Higher-order language requirements*

1. *Signal handshake protocols (for example, XON-XOFF, ACK-NACK)*
2. *Reliability requirements*

*(10) Criticality of the application*

*(11) Safety and security considerations*

*This section captures non-functional requirements in the customers language. A more formal presentation of these will occur in section 3.*

## 2.5 Assumptions and Dependencies

*List each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption might be that a specific operating system would be available on the hardware designated for the software product. If, in fact, the operating system were not available, the SRS would then have to change accordingly.*

*This section is catch-all for everything else that might influence the design of the system and that did not fit in any of the categories above.*

## 2.6 Apportioning of Requirements.

*Identify requirements that may be delayed until future versions of the system. After you look at the project plan and hours available, you may realize that you just cannot get everything done. This section divides the requirements into different sections for development and delivery. Remember to check with the customer – they should prioritize the requirements and decide what does and does not get done. This can also be useful if you are using an iterative life cycle model to specify which requirements will map to which interation.*

# 3. Specific Requirements

*This section contains all the software requirements at a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system and all functions performed by the system in response to an input or in support of an output. The following principles apply:*

1. *Specific requirements should be stated with all the characteristics of a good SRS*
2. *correct*
3. *unambiguous*
4. *complete*
5. *consistent*
6. *ranked for importance and/or stability*
7. *verifiable*
8. *modifiable*
9. *traceable*
10. *Specific requirements should be cross-referenced to earlier documents that relate*
11. *All requirements should be uniquely identifiable (usually via numbering like 3.1.2.3)*
12. *Careful attention should be given to organizing the requirements to maximize readability (Several alternative organizations are given at end of document)*

*Before examining specific ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in the following subclasses. This section reiterates section 2, but is for developers not the customer. The customer buys in with section 2, the designers use section 3 to design and build the actual application.*

*Remember this is not design. Do not require specific software packages, etc unless the customer specifically requires them. Avoid over-constraining your design. Use proper terminology:*

*The system shall… A required, must have feature*

*The system should… A desired feature, but may be deferred til later*

*The system may… An optional, nice-to-have feature that may never make it to implementation.*

*Each requirement should be uniquely identified for traceability. Usually, they are numbered 3.1, 3.1.1, 3.1.2.1 etc. Each requirement should also be testable. Avoid imprecise statements like, “The system shall be easy to use” Well no kidding, what does that mean? Avoid “motherhood and apple pie” type statements, “The system shall be developed using good software engineering practice”*

*Avoid examples, This is a specification, a designer should be able to read this spec and build the system without bothering the customer again. Don’t say things like, “The system shall accept configuration information such as name and address.” The designer doesn’t know if that is the only two data elements or if there are 200. List every piece of information that is required so the designers can build the right UI and data tables.*

## 

## 3.1 External Interfaces

3.1.1 GUI Specifications

3.1.1.1 The GUI shall contain multiple graphs on screen that represent the application of different control algorithms to the model

3.1.1.2 The GUI shall contain buttons for each control algorithm (if applicable) that provide options to modify salient controls parameters (ex. Gain for a PID alg.)

3.1.1.3 The GUI shall provide an interface for the input of a model., parameters, and parameter confidences.

3.1.1.4 The GUI shall provide a button that enable the re-running of a simulation.

3.1.1.5 The GUI shall provide a button to start a simulation

3.1.1.6 The GUI shall provide an attention-grabbing way to display errors that bubble up from the back-end.

3.1.1.7 The GUI should provide basic error-checking services itself.

3.1.1.8 The GUI should provide commonly-used models (ex. First-order, second-order, etc.).

*This contains a detailed description of all inputs into and outputs from the software system. It complements the interface descriptions in section 2 but does not repeat information there. Remember section 2 presents information oriented to the customer/user while section 3 is oriented to the developer.*

*It contains both content and format as follows:*

1. *Name of item*
2. *Description of purpose*
3. *Source of input or destination of output*
4. *Valid range, accuracy and/or tolerance*
5. *Units of measure*
6. *Timing*
7. *Relationships to other inputs/outputs*
8. *Screen formats/organization*
9. *Window formats/organization*
10. *Data formats*
11. *Command formats*
12. *End messages*

## 3.2 Functions

3.2.1 I/O Functions

3.2.1.1 The system shall check all user input for common security vulnerabilities.

3.2.1.2 The system shall provide a response to each user GET request.

3.2.1.3 The system shall return a set of points back:

3.2.1.4 The system shall provide an error message and error code for any Errors or invalid user input

3.2.2 Model application functions

3.2.2.1 The system shall numerically evaluate the function utilizing the confidence intervals defined by the parameters.

3.2.2.2 The system shall transform the numerical evaluation to the LaPlace domain.

3.2.2.3 The system shall evaluate the control algorithm on a standard control loop (in, controller, model, out, repeat)

3.2.2.4 The system shall make use of deviation variables

3

*Functional requirements define the fundamental 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:*

1. *Validity checks on the inputs*
2. *Exact sequence of operations*
3. *Responses to abnormal situation, including*
4. *Overflow*
5. *Communication facilities*
6. *Error handling and recovery*
7. *Effect of parameters*
8. *Relationship of outputs to inputs, including*
9. *Input/Output sequences*
10. *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.*

## 3.3 Performance Requirements

*This subsection specifies 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:*

*(a) The number of terminals to be supported*

*(b) The number of simultaneous users to be supported*

*(c) Amount and type of information to be handled*

*Static numerical requirements are sometimes identified under a separate section entitled capacity.*

*Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.*

*All of these requirements should be stated in measurable terms.*

*For example,*

*95% of the transactions shall be processed in less than 1 second*

*rather than,*

*An operator shall not have to wait for the transaction to complete.*

*(Note: Numerical limits applied to one specific function are normally specified as part of the processing subparagraph description of that function.)*

## 3.4 Logical Database Requirements

*This section specifies the logical requirements for any information that is to be placed into a database. This may include:*

1. *Types of information used by various functions*
2. *Frequency of use*
3. *Accessing capabilities*
4. *Data entities and their relationships*
5. *Integrity constraints*
6. *Data retention requirements*

*If the customer provided you with data models, those can be presented here. ER diagrams (or static class diagrams) can be useful here to show complex data relationships. Remember a diagram is worth a thousand words of confusing text.*

## 3.5 Design Constraints

*Specify design constraints that can be imposed by other standards, hardware limitations, etc.*

### 3.5.1 Standards Compliance

*Specify the requirements derived from existing standards or regulations. They might include:*

*(1) Report format*

*(2) Data naming*

*(3) Accounting procedures*

*(4) Audit Tracing*

*For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.*

## 3.6 Software System Attributes

*There are a number of attributes of software that can serve as requirements. It is important that required attributes by specified so that their achievement can be objectively verified. The following items provide a partial list of examples. These are also known as non-functional requirements or quality attributes.*

*These are characteristics the system must possess, but that pervade (or cross-cut) the design. These requirements have to be testable just like the functional requirements. Its easy to start philosophizing here, but keep it specific.*

### 3.6.1 Reliability

*Specify the factors required to establish the required reliability of the software system at time of delivery. If you have MTBF requirements, express them here. This doesn’t refer to just having a program that does not crash. This has a specific engineering meaning.*

### 3.6.2 Availability

*Specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart. This is somewhat related to reliability. Some systems run only infrequently on-demand (like MS Word). Some systems have to run 24/7 (like an e-commerce web site). The required availability will greatly impact the design. What are the requirements for system recovery from a failure? “The system shall allow users to restart the application after failure with the loss of at most 12 characters of input”.*

### 3.6.3 Security

*Specify the factors that would protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to:*

1. *Utilize certain cryptographic techniques*
2. *Keep specific log or history data sets*
3. *Assign certain functions to different modules*
4. *Restrict communications between some areas of the program*
5. *Check data integrity for critical variables*

### 3.6.4 Maintainability

*Specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices. If someone else will maintain the system*

### 3.6.5 Portability

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

1. *Percentage of components with host-dependent code*
2. *Percentage of code that is host dependent*
3. *Use of a proven portable language*
4. *Use of a particular compiler or language subset*
5. *Use of a particular operating system*

*Once the relevant characteristics are selected, a subsection should be written for each, explaining the rationale for including this characteristic and how it will be tested and measured. A chart like this might be used to identify the key characteristics (rating them High or Medium), then identifying which are preferred when trading off design or implementation decisions (with the ID of the preferred one indicated in the chart to the right). The chart below is optional (it can be confusing) and is for demonstrating tradeoff analysis between different non-functional requirements. H/M/L is the relative priority of that non-functional requirement.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Characteristic** | **H/M/L** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| 1 | Correctness |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | Efficiency |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | Flexibility |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | Integrity/Security |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | Interoperability |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 | Maintainability |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 | Portability |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 | Reliability |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 | Reusability |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 | Testability |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 11 | Usability |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 12 | Availability |  |  |  |  |  |  |  |  |  |  |  |  |  |

*Definitions of the quality characteristics not defined in the paragraphs above follow.*

*• Correctness - extent to which program satisfies specifications, fulfills user’s mission objectives*

*• Efficiency - amount of computing resources and code required to perform function*

*• Flexibility - effort needed to modify operational program*

*• Interoperability - effort needed to couple one system with another*

*• Reliability - extent to which program performs with required precision*

*• Reusability - extent to which it can be reused in another application*

*• Testability - effort needed to test to ensure performs as intended*

*• Usability - effort required to learn, operate, prepare input, and interpret output*

*THE FOLLOWING (3.7) is not really a section, it is talking about how to organize requirements you write in section 3.2. At the end of this template there are a bunch of alternative organizations for section 3.2. Choose the ONE best for the system you are writing the requirements for.*

## 3.7 Organizing the Specific Requirements

*For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organizing these in a manner optimal for understanding. There is no one optimal organization for all systems. Different classes of systems lend themselves to different organizations of requirements in section 3. Some of these organizations are described in the following subclasses.*

### 3.7.1 System Mode

*Some systems behave quite differently depending on the mode of operation. When organizing by mode there are two possible outlines. The choice depends on whether interfaces and performance are dependent on mode.*

### 3.7.2 User Class

*Some systems provide different sets of functions to different classes of users.*

### 3.7.3 Objects

*Objects are real-world entities that have a counterpart within the system. Associated with each object is a set of attributes and functions. These functions are also called services, methods, or processes. Note that sets of objects may share attributes and services. These are grouped together as classes.*

### 3.7.4 Feature

*A feature is an externally desired service by the system that may require a sequence of inputs to effect the desired result. Each feature is generally described in as sequence eof stimulus-response pairs.*

### 3.7.5 Stimulus

*Some systems can be best organized by describing their functions in terms of stimuli.*

### 3. 7.6 Response

*Some systems can be best organized by describing their functions in support of the generation of a response.*

### 3.7.7 Functional Hierarchy

*When none of he above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by either common inputs, common outputs, or common internal data access. Data flow diagrams and data dictionaries can be use dot show the relationships between and among the functions and data.*

## 3.8 Additional Comments

*Whenever a new SRS is contemplated, more than one of the organizational techniques given in 3.7 may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification.*

*Three are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, finite state machines or state charts may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.*

*In any of the outlines below, those sections called “Functional Requirement i” may be described in native language, in pseudocode, in a system definition language, or in four subsections titled: Introduction, Inputs, Processing, Outputs.*

# Change Management Process

*Identify the change management process to be used to identify, log, evaluate, and update the SRS to reflect changes in project scope and requirements. How are you going to control changes to the requirements. Can the customer just call up and ask for something new? Does your team have to reach consensus? How do changes to requirements get submitted to the team? Formally in writing, email or phone call?*

# Document Approvals

*Identify the approvers of the SRS document. Approver name, signature, and date should be used.*

# Supporting Information

*The supporting information makes the SRS easier to use. It includes:*

1. *Table of Contents*
2. *Index*
3. *Appendices*

*The Appendices are not always considered part of the actual requirements specification and are not always necessary. They may include:*

*(a) Sample I/O formats, descriptions of cost analysis studies, results of user surveys*

*(b) Supporting or background information that can help the readers of the SRS*

*(c) A description of the problems to be solved by the software*

*(d) Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements*

*When Appendices are included, the SRS should explicitly state whether or not the Appendices are to be considered part of the requirements.*

Tables on the following pages provide alternate ways to structure section 3 on the specific requirements. You should pick the best one of these to organize section 3 requirements.

**Outline for SRS Section 3**

**Organized by mode: Version 1**

3. Specific Requirements

3.1 External interface requirements

1. User interfaces
2. Hardware interfaces
3. Software interfaces
4. Communications interfaces
5. Functional requirements

3.2.1 Mode 1

3.2.1.1 Functional requirement 1.1

.....

3.2.1.*n* Functional requirement 1.*n*

1. Mode 2

.....

3.2.*m* Mode *m*

3.2.*m*.1 Functional requirement *m*.1

.....

3.2.*m.n* Functional requirement *m.n*

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software system attributes

3.6 Other requirements

**Outline for SRS Section 3**

**Organized by mode: Version 2**

3. Specific Requirements

3.1 Functional Requirements

1. Mode 1

3.1.1.1 External interfaces

3.1.1.1 User interfaces

3.1.1.2 Hardware interfaces

3.1.1.3 Software interfaces

3.1.1.4 Communications interfaces

3.1.1.2 Functional Requirement

3.1.1.2.1 Functional requirement 1

.....

3.1.1.2.*n* Functional requirement *n*

3.1.1.3 Performance

3.1.2 Mode 2

.....

3.1.*m* Mode *m*

1. Design constraints
2. Software system attributes
3. Other requirements

**Outline for SRS Section 3**

**Organized by user class (i.e. different types of users ->System Adminstrators, Managers, Clerks, etc.)**

3. Specific Requirements

3.1 External interface requirements

1. User interfaces
2. Hardware interfaces
3. Software interfaces
4. Communications interfaces
5. Functional requirements

3.2.1 User class 1

3.2.1.1 Functional requirement 1.1

.....

3.2.1.*n* Functional requirement 1.*n*

1. User class 2

.....

3.2.*m* User class *m*

3.2.*m*.1 Functional requirement *m*.1

.....

3.2.*m.n* Functional requirement *m.n*

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software system attributes

3.6 Other requirements

**Outline for SRS Section 3**

**Organized by object (Good if you did an object-oriented analysis as part of your requirements)**

3 Specific Requirements

3.1 External interface requirements

1. User interfaces
2. Hardware interfaces
3. Software interfaces
4. Communications interfaces
5. Classes/Objects

3.2.1 Class/Object 1

3.2.1.1 Attributes (direct or inherited)

1. Attribute 1

.....

3.2.1.1.*n* Attribute *n*

1. Functions (services, methods, direct or inherited)

3.2.1.2.1 Functional requirement 1.1

.....

3.2.1.2.*m* Functional requirement 1.*m*

3.2.1.3 Messages (communications received or sent)

3.2.2 Class/Object 2

.....

3.2.*p* Class/Object *p*

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software system attributes

3.6 Other requirements

**Outline for SRS Section 3**

**Organized by feature (Good when there are clearly delimited feature sets.**

3 Specific Requirements

3.1 External interface requirements

1. User interfaces
2. Hardware interfaces
3. Software interfaces
4. Communications interfaces
5. System features

3.2.1 System Feature 1

3.2.1.1 Introduction/Purpose of feature

3.2.1.2 Stimulus/Response sequence

3.2.1.3 Associated functional requirements

3.2.1.3.1 Functional requirement 1

.....

3.2.1.3.*n* Functional requirement *n*

3.2.2 System Feature 2

.....

3.2.*m* System Feature *m*

.....

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software system attributes

3.6 Other requirements

**Outline for SRS Section 3**

**Organized by stimulus (Good for event driven systems where the events form logical groupings)**

3 Specific Requirements

3.1 External interface requirements

1. User interfaces
2. Hardware interfaces
3. Software interfaces
4. Communications interfaces
5. Functional requirements

3.2.1 Stimulus 1

3.2.1.1 Functional requirement 1.1

.....

3.2.1.*n* Functional requirement 1.*n*

3.2.2 Stimulus 2

.....

3.2.*m* Stimulus *m*

3.2.*m*.1 Functional requirement *m*.1

.....

3.2.*m.n* Functional requirement *m.n*

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software system attributes

3.6 Other requirements

**Outline for SRS Section 3**

**Organized by response (Good for event driven systems where the responses form logical groupings)**

3 Specific Requirements

3.1 External interface requirements

1. User interfaces
2. Hardware interfaces
3. Software interfaces
4. Communications interfaces
5. Functional requirements

3.2.1 Response 1

3.2.1.1 Functional requirement 1.1

.....

3.2.1.*n* Functional requirement 1.*n*

3.2.2 Response 2

.....

3.2.*m* Response *m*

3.2.*m*.1 Functional requirement *m*.1

.....

3.2.*m.n* Functional requirement *m.n*

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software system attributes

3.6 Other requirements

**Outline for SRS Section 3**

**Organized by functional hierarchy (Good if you have done structured analysis as part of your design.)**

3 Specific Requirements

3.1 External interface requirements

1. User interfaces
2. Hardware interfaces
3. Software interfaces
4. Communications interfaces
5. Functional requirements

3.2.1 Information flows

3.2.1.1 Data flow diagram 1

1. Data entities
2. Pertinent processes
3. Topology

3.2.1.2 Data flow diagram 2

1. Data entities
2. Pertinent processes
3. Topology

.....

3.2.1.*n* Data flow diagram *n*

3.2.1.*n*.1 Data entities

3.2.1.*n*.2 Pertinent processes

3.2.1.*n*.3 Topology

3.2.2 Process descriptions

1. Process 1
2. Input data entities
3. Algorithm or formula of process
4. Affected data entities

3.2.2.2 Process 2

3.2.2.2.1 Input data entities

3.2.2.2.2 Algorithm or formula of process

3.2.2.2.3 Affected data entities

.….

3.2.2.*m* Process *m*

3.2.2.*m*.1 Input data entities

3.2.2.*m*.2 Algorithm or formula of process

3.2.2.*m*.3 Affected data entities

3.2.3 Data construct specifications

3.2.3.1 Construct 1

3.2.3.1.1 Record type

3.2.3.1.2 Constituent fields

3.2.3.2 Construct 2

3.2.3.2.1 Record type

3.2.3.2.2 Constituent fields

…..

3.2.3.*p* Construct *p*

3.2.3.*p*.1 Record type

3.2.3.*p*.2 Constituent fields

3.2.4 Data dictionary

3.2.4.1 Data element 1

3.2.4.1.1 Name

3.2.4.1.2 Representation

3.2.4.1.3 Units/Format

3.2.4.1.4 Precision/Accuracy

3.2.4.1.5 Range

3.2.4.2 Data element 2

3.2.4.2.1 Name

3.2.4.2.2 Representation

3.2.4.2.3 Units/Format

3.2.4.2.4 Precision/Accuracy

3.2.4.2.5 Range

…..

3.2.4.*q* Data element *q*

3.2.4.*q*.1 Name

3.2.4.*q*.2 Representation

3.2.4.*q*.3 Units/Format

3.2.4.*q*.4 Precision/Accuracy

3.2.4.*q*.5 Range

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software system attributes

3.6 Other requirements

**Outline for SRS Section 3**

**Showing multiple organizations (Can’t decide? Then glob it all together)**

3 Specific Requirements

3.1 External interface requirements

1. User interfaces
2. Hardware interfaces
3. Software interfaces
4. Communications interfaces
5. Functional requirements

3.2.1 User class 1

3.2.1.1 Feature 1.1

3.2.1.1.1 Introduction/Purpose of feature

3.2.1.1.2 Stimulus/Response sequence

3.2.1.1.3 Associated functional requirements

3.2.1.2 Feature 1.2

3.2.1.2.1 Introduction/Purpose of feature

3.2.1.2.2 Stimulus/Response sequence

3.2.1.2.3 Associated functional requirements

…..

3.2.1.*m* Feature 1.*m*

3.2.1.*m*.1 Introduction/Purpose of feature

3.2.1.*m*.2 Stimulus/Response sequence

3.2.1.*m*.3 Associated functional requirements

3.2.2 User class 2

.....

3.2.*n* User class *n*

.....

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software system attributes

3.6 Other requirements

**Outline for SRS Section 3**

**Organized by Use Case (Good when following UML development)**

3. Specific Requirements

3.1 External Actor Descriptions

3.1.1 Human Actors

3.1.2 Hardware Actors

3.1.3 Software System Actors

3.2 Use Case Descriptions

3.2.1 Use Case 1

3.2.2 Use Case 2

3.2.n Use Case n

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software system attributes

3.6 Other requirements