**91.411.201: Software Engineering I**

**Software Requirements Specification (SRS) Template**

Items that are intended to stay in as part of your document are in **bold**; explanatory comments are in *italic* text. Plain text is used where you might insert wording about your project.

The document in this file is an annotated outline for specifying software requirements, adapted from the IEEE Guide to Software Requirements Specifications (Std 830-1993).

Tailor this to your needs, removing explanatory comments as you go along. Where you decide to omit a section, you might keep the header, but insert a comment saying why you omit the data.

**91.411.201: Software Engineering I**

**Bubble Warriors Adventure**

**Software Requirements Specification**

**Document**

**Version: (1)** **Date: (04/04/2007)**

**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 is to build a 2-D RPG styled video game for Windows and Unix environments.

## 1.2 Scope

The software being produced is called “Bubble Warriors Adventure”. Will have a storyline, controllable character, player vs enemy A.I. battling, obtainable items, and conversation with non-player characters, leveling up, and upgradeable statistics for the player. Will not feature multiplayer option.

*In this subsection:*

1. *Identify the software product(s) to be produced by name*
2. *Explain what the software product(s) will, and, if necessary, will not do*
3. *Describe the application of the software being specified, including relevant benefits, objectives, and goals*
4. *Be consistent with similar statements in higher-level specifications if they exist*

*This should be an executive-level summary. Do not enumerate the whole requirements list here.*

## 1.3 Definitions, Acronyms, and Abbreviations.

1. Bubble Warriors Adventure – BWA
2. Bubble Warriors Adventure – “this product”

## 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

The SRS contains specific specifications on what this game aims to accomplish, and the interactivity with the user.

*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

*Describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in section 3, and makes them easier to understand*. *In a sense, this section tells the requirements in plain English for the consumption of the customer. Section3 will contain a specification written for the developers.*

## 2.1 Product Perspective

This product is a self-contained product. So long as the user has the product on their system, will be able to be run.

### 2.1.1 System Interfaces

### This product has no system interfaces.

### 2.1.2 Interfaces

This product will have a standalone window, with a GUI for the menus, and an overlay GUI for the main gameplay. This is meant for any user which can use two hands effectively, and have an interest for this style of video game. A keyboard and mouse is required, or equivalent.

### 2.1.3 Hardware Interfaces

### This system has no hardware interface requirements. A basic, single Windows or Unix environment computer will be enough to run this program.

### 2.1.4 Software Interfaces

*Specify the use of other required software products and interfaces with other application systems. For each required software product, include:*

1. *Name*
2. *Mnemonic*
3. *Specification number*
4. *Version number*
5. *Source*

*For each interface, provide:*

1. *Discussion of the purpose of the interfacing software as related to this software product*
2. *Definition of the interface in terms of message content and format*

*Here we document the APIs, versions of software that we do not have to write, but that our system has to use. For instance if your customer uses SQL Server 7 and you are required to use that, then you need to specify i.e.*

*2.1.4.1 Microsoft SQL Server 7. The system must use SQL Server as its database component. Communication with the DB is through ODBC connections. The system must provide SQL data table definintions to be provided to the company DBA for setup.*

*A key point to remember is that you do NOT want to specify software here that you think would be good to use. This is only for* ***customer-specified systems*** *that you* ***have*** *to interact with. Choosing SQL Server 7 as a DB without a customer requirement is a Design choice, not a requirement. This is a subtle but important point to writing good requirements and not over-constraining the design.*

### 2.1.5 Communications Interfaces

There are no communication interfaces or protocols in this system. Will be dependent on a single local system.

### 2.1.6 Memory Constraints

There are no memory constraints for this system, but will optimize memory usage as much as we can.

### 2.1.7 Operations

BWA will be played using both the keyboard and mouse, where the keyboard is for movement and interaction, while the mouse is mainly for GUI elements. The user would remain in the game for the entire operation, and would have the option to pause gameplay in order to leave the operation. The user can also save the current state of the system and be able to come back to it at a later date.

### 2.1.8 Site Adaptation Requirements

There is no need for any site adaptation, as the nature of the system is not of any critical importance.

## 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

*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

*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 product has no database, and all data saved will be encoded with Python.

## 3.5 Design Constraints

This product has no design constraints imposed by other standards or hardware limitations.

### 3.5.1 Standards Compliance

Uniform coding style is written on the github readme. No other standards or regulations apply. All data is not sensitive, and this product’s purpose is for entertainment.

## 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

This product has a standard availability. If the product crashes, The data will revert to the most recent save of the game. This will be intentional, as some video games people play would prefer to exit without saving if something game-breaking bad occurs.

### 3.6.3 Security

All save data and certain data storage files will be encrypted using Python. If a file has an error, will attempt to revert back to the default file config.

### 3.6.4 Maintainability

All coding styles and descriptions of all code are maintained on the GitHub repository, and anyone can modify the code based on the coding style.

### 3.6.5 Portability

This program is very portable. Any Windows or Unix environment can run it. If there is an OS or machine which will not run the game, the user can compile the necessary libraries and create their own executable tailored to their system’s compiler.

*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

All user types will have the same functionality, as there is only one kind of user class.

### 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