EECS 348 Group Project

Software Requirements Specifications

Version 1.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 03/08/2024 | 1.0 | Added section 4: Classification of Functional Requirements | Katie Nordberg |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 4

1.5 Overview 4

2. Overall Description 5

2.1 Product perspective 5

2.1.1 System Interfaces 5

2.1.2 User Interfaces 5

2.1.3 Hardware Interfaces 5

2.1.4 Software Interfaces 5

2.1.5 Communication Interfaces 5

2.1.6 Memory Constraints 5

2.1.7 Operations 5

2.2 Product functions 5

2.3 User characteristics 5

2.4 Constraints 5

2.5 Assumptions and dependencies 5

2.6 Requirements subsets 5

3. Specific Requirements 5

3.1 Functionality 5

3.1.1 <Functional Requirement One> 6

3.2 Use-Case Specifications 6

3.3 Supplementary Requirements 6

4. Classification of Functional Requirements 6

5. Appendices 6

Software Requirements Specifications

# Introduction

[The introduction of the **Software Requirements Specification (SRS)** provides an overview of the entire **SRS**. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the **SRS**.]

[Note: The **SRS** captures the complete software requirements for the system, or a portion of the system. Following is a typical **SRS** outline for a project **using use-case modeling**. This artifact consists of a package containing use cases of the use-case model and applicable Supplementary Specifications and other supporting information.]

[Many different arrangements of an **SRS** are possible. Refer to [IEEE830-1998] for further elaboration of these explanations, as well as other options for **SRS** organization.]

## Purpose

[Specify the purpose of this **SRS**. The **SRS** fully describes the external behavior of the application or subsystem identified. It also describes nonfunctional requirements, design constraints, and other factors necessary to provide a complete and comprehensive description of the requirements for the software.]

## Scope

[A brief description of the software application that the **SRS** applies to, the feature or other subsystem grouping, what Use-Case model(s) it is associated with, and anything else that is affected or influenced by this document.]

## Definitions, Acronyms, and Abbreviations

[This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the **SRS**. This information may be provided by reference to the project’s Glossary.]

## References

[This subsection provides a complete list of all documents referenced elsewhere in the **SRS**. Identify each document by title, report number if applicable, date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

## Overview

[This subsection describes what the rest of the **SRS** contains and explains how the document is organized.]

# Overall Description

[This section of the **SRS** describes 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 detail in Section 3, and makes them easier to understand. Include such items as:

## Product perspective

### System Interfaces

### User Interfaces

### Hardware Interfaces

### Software Interfaces

### Communication Interfaces

### Memory Constraints

### Operations

## Product functions

## User characteristics

## Constraints

## Assumptions and dependencies

## Requirements subsets

# Specific Requirements

[This section of the **SRS** contains all software requirements to 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. When using use-case modeling, these requirements are captured in the Use Cases and the applicable supplementary specifications. If use-case modeling is not used, the outline for supplementary specifications may be inserted directly into this section, as shown below.]

## Functionality

[This section describes the functional requirements of the system for those requirements that are expressed in the natural language style. For many applications, this may constitute the bulk of the **SRS** package and thought should be given to the organization of this section. This section is typically organized by feature, but alternative organization methods may also be appropriate; for example, organization by user or organization by subsystem. Functional requirements may include feature sets, capabilities, and security.

Where application development tools, such as requirements tools, modeling tools, and the like, are employed to capture the functionality, this section of the document would refer to the availability of that data, indicating the location and name of the tool used to capture the data.]

### <Functional Requirement One>

[The requirement description.]

## Use-Case Specifications

[In use-case modeling, the use cases often define the majority of the functional requirements of the system, along with some non-functional requirements.]

## Supplementary Requirements

[Supplementary Specifications capture other requirements, e.g., non-functional requirements and development constraints, that are not included in the use cases and non-functional requirements.]

# Classification of Functional Requirements

|  |  |
| --- | --- |
| **Functionality** | **Type** |
| 1. Operator Support 2. AND (&) Support: n & k Returns True if both n and k are True. Returns False otherwise. 3. OR (|) Support: n | k Returns True if at least one of n or k are True. Returns False otherwise. 4. NOT (!) Support: !n Returns the opposite Boolean value of n. 5. NAND (@) Support: n @ k Returns True if both n and k are False. Returns False otherwise. 6. XOR ($) Support: n $ k Returns True if only one of n or k are True. Returns False otherwise. | Essential |
| 1. Expression Parsing: Parse Boolean expressions entered by the user. Expressions will be given in infix notation. (Infix notation is the common way of writing mathematical expressions.) These expressions will be interpreted respecting operator precedence and parentheses. This requirement is not to evaluate the expression (that is requirement 4). Rather, this requirement describes the need to take in a user inputted expression and determine which operations and truth values must be used. | Essential |
| 1. Truth Value Input: Allow users to define truth values for each variable. Users will be able to input T or F to represent True or False respectively. These values will be used to define the Boolean value of variables which will then be used in expressions to be evaluated. | Essential |
| 1. Evaluation and Output: Evaluate expressions according to the operator precedence defined by feature. Then display the Boolean result of the entire expression.   There is no single defined precedence for logical operators. Each language defines their own precedence. After researching many ways, we will use the following precedence:  Parentheses first to be evaluated  NOT (!)  XOR ($), NAND (@) Evaluated from left to right in the expression.  AND (&)  OR (|) last to be evaluated | Essential |
| 1. Error Handling: Provide a helpful and descriptive message to the user when an error occurs. After an error is discovered and the user is informed, the program will prompt the user to re-enter the expression. This cycle will continue until a valid expression is entered such that it is properly evaluated.   The following errors must be handled by the program without crashing.   1. Missing Parentheses 2. Unknown characters (i.e. letters or symbols that carry no meaning in our program. This may be undeclared variables, or unsupported operators. We make no distinction between these as there is no way to tell the difference between an unknown operand and an unknown operator because while the user may enter an unknown operator, the program could interpret it as an unknown operand with a missing operator or vice versa. Therefore, we will keep the message simple and group unknown operands and unknown operators into one error.) 3. Missing Operands 4. Empty Expression | Essential |
| 1. Parentheses Handling: Expressions should be evaluated with respect to the most nested expression first and the least nested expression last. This feature requires that our program support parentheses (including nested parentheses) in user-entered expressions. | Essential |
| 1. Display a truth table showing the values of each input, each intermediate expression, and the final expression. This is marked as desirable because it is not a requirement of the project. However, it should not be extremely difficult to implement, and it would earn the project extra credit. | Desirable |

# Appendices

[When appendices are included, the **SRS** should explicitly state whether or not the appendices are to be considered part of the requirements]