Ice Cream Manager [ICM]

Software Requirements Specification Document

Version No. v0.2

Project Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Revision Author | Description of Revision |
| 0.1 | 2016-2-19 | Cosmosys Team | Initial content generation. |
| 0.2 | 2016-2-20 | Marc King | Added Specificity, Completeness, and Responsibility metrics for the Analysis Metrics section (8.3). |
| 0.3 | 2016-3-3 | Rodney Lewis | Added draft of sections 3.1, 3.1.2, 3.1.3 |

Table of Contents

[1.0 Introduction 5](#_Toc443744660)

[1.1 Goals and Objectives 5](#_Toc443744661)

[1.2 Statement of Scope 5](#_Toc443744662)

[1.3 Software Context 5](#_Toc443744663)

[1.4 Major Constraints 5](#_Toc443744664)

[2.0 Usage Scenario 5](#_Toc443744665)

[2.1 User Profiles 5](#_Toc443744666)

[2.2 Use-Cases 5](#_Toc443744667)

[2.3 Special Usage Considerations 5](#_Toc443744668)

[3.0 Data Model and Description 5](#_Toc443744669)

[3.1 Data Description 6](#_Toc443744670)

[3.1.1 Data Objects 6](#_Toc443744671)

[3.1.2 Relationships 6](#_Toc443744672)

[3.1.3 Complete Data Model 6](#_Toc443744673)

[3.1.4 Data Dictionary 6](#_Toc443744674)

[4.0 Functional Model and Description 6](#_Toc443744675)

[4.1 Use Cases 6](#_Toc443744676)

[4.2 Software Interface Description 6](#_Toc443744677)

[4.2.1 External Machine Interfaces 6](#_Toc443744678)

[4.2.2 External System Interfaces 6](#_Toc443744679)

[4.2.3 Human Interface 7](#_Toc443744680)

[4.3 Sequence Diagrams 7](#_Toc443744681)

[4.4 Communication Diagrams 7](#_Toc443744682)

[5.0 Behavioral Model and Description 7](#_Toc443744683)

[5.1 Description for Software Behavior 7](#_Toc443744684)

[5.1.1 Events 7](#_Toc443744685)

[5.1.2 States 7](#_Toc443744686)

[5.2 State Transition Diagrams 7](#_Toc443744687)

[6.0 Restrictions, Limitations, and Constraints 7](#_Toc443744688)

[7.0 Validation Criteria 7](#_Toc443744689)

[7.1 Classes of Tests 7](#_Toc443744690)

[7.2 Expected Software Response 8](#_Toc443744691)

[7.3 Performance Bounds 8](#_Toc443744692)

[8.0 Appendices 8](#_Toc443744693)

[8.1 System Traceability Matrix 8](#_Toc443744694)

[8.2 Product Strategies 8](#_Toc443744695)

[8.3 Analysis Metrics to be Used 8](#_Toc443744696)

[8.3.1 Goal of the Metrics 8](#_Toc443744697)

[8.3.2 Requirements Model Metrics 8](#_Toc443744698)

[8.3.3 Object-Oriented Model Metrics 9](#_Toc443744699)

[8.3.4 Source Code Quality Metrics 9](#_Toc443744700)

[8.4 Supplementary Information (as required) 9](#_Toc443744701)

[9.0 Software Requirements Specification Review and Signoff 10](#_Toc443744702)

# 1.0 Introduction

This section provides an overview of the entire requirement document. This document describes all data, functional and behavioral requirements for software.

## 1.1 Goals and Objectives

Overall goals and software objectives are described.

## 1.2 Statement of Scope

A description of the software is presented. Major inputs, processing functionality and outputs are described without regard to implementation detail.

## 1.3 Software Context

The software is placed in a business or product line context. Strategic issues relevant to context are discussed. The intent is for the reader to understand the 'big picture'.

## 1.4 Major Constraints

Any business or product line constraints that will impact the manner in which the software is to be specified, designed, implemented or tested are noted here.

# 2.0 Usage Scenario

This section provides a usage scenario for the software. It organized information collected during requirements elicitation into use-cases.

## 2.1 User Profiles

The profiles of all user categories are described here.

## 2.2 Use-Cases

LIST of all use-cases for the software are presented.

## 2.3 Special Usage Considerations

Special requirements associated with the use of the software are presented.

# 3.0 Data Model and Description

This section describes information domain for the software

## 3.1 Data Description

|  |  |
| --- | --- |
| **Data Object** | **Description** |
| Item | A unit of ice cream |
| Sale | The result of an item being sold |
| Route | A collection or ordered zones |
| Zone | A location consisting of a city and a subset of the city that is mapped to a number |
| Truck | Object that holds items and travels routes |
| Poll | A collection of votes that are used to make changes to future inventory |
| Driver | Object that uses truck and drives routes |
| Preset | A pre-determined inventory created by the user |

### 3.1.1 Data Objects

* Item
  + Type: The price and name of a unit of ice cream
  + Item number: Unique reference number to an item
  + Expiration date: Date used to determine the freshness of a unit of ice cream
* Sale
  + Sale number: Unique reference number of a sale
  + Sale date: Date that the sale was made
* Route
  + Route number: Unique reference number of a route
* Zone
  + Zone number: A subset of a city that is mapped to a number
  + City: Name of a location
  + Distance: How long a zone is in miles
* Truck
  + Truck number: Unique reference number to a truck
  + Item capacity: The total number of items that a truck can have in its inventory
  + Fuel capacity: The total amount of fuel, in gallons, that a truck can hold
  + Fuel amount: The remaining amount of fuel, in gallons, that a truck has
  + Fuel consumption: Fuel efficiency in miles per gallon
* Poll
  + Poll number: Unique reference number for a poll
  + Poll name: User defined name for a poll
  + Total votes: The number of votes that were processed for a poll
* Driver
  + Driver number: Unique reference number for a driver
  + Driver name: Name of a driver
* Preset
  + Preset number: Unique reference number for a preset
  + Preset name: User defined name for a preset
  + Start date: The date that a preset will take effect
  + End date: The date that a preset will no longer be in effect

### 3.1.2 Relationships

|  |  |
| --- | --- |
| **Item** | |
| **Responsibility** | **Collaborator** |
| Add items |  |
| Delete items |  |
| Set expiration date for items |  |
| Set item type |  |

|  |  |
| --- | --- |
| **Sale** | |
| **Responsibility** | **Collaborator** |
| Track item sales | Item |
| Calculate sales based on inventory | Item |

|  |  |
| --- | --- |
| **Route** | |
| **Responsibility** | **Collaborator** |
| Display assigned truck | Truck |
| Use inventory presets | Presets |
| Define sales by route | Sale |
| Display route composition | Zone |

|  |  |
| --- | --- |
| **Zone** | |
| **Responsibility** | **Collaborator** |
| Edit zone |  |

|  |  |
| --- | --- |
| **Truck** | |
| **Responsibility** | **Collaborator** |
| Define sales by truck | Sale |
| Display assigned route | Route |
| Calculate fuel efficiency | Zone |
| Assign route | Route |
| Assign driver | Driver |
| Use inventory preset | Preset |
| Calculate fuel usage |  |

|  |  |
| --- | --- |
| **Poll** | |
| **Responsibility** | **Collaborator** |
| Create poll | Zone |
| View results of poll | Zone |

|  |  |
| --- | --- |
| **Driver** | |
| **Responsibility** | **Collaborator** |
| Define sales by driver | Sale |
|  |  |

|  |  |
| --- | --- |
| **Preset** | |
| **Responsibility** | **Collaborator** |
| Create Preset | Item |
| Apply Preset | Truck, Route |
| Modify Preset | Item |

### 3.1.3 Complete Data Model

An UML Class model for the software is developed

### 3.1.4 Data Dictionary

A reference to the data dictionary is provided. The dictionary is maintained in electronic form.

# 4.0 Functional Model and Description

Description of major software functions along with UML Use Case, sequence, and communication diagrams.

## 4.1 Use Cases

A detailed description of each software function is presented by completing a use case template.

Cross reference this document with file name of use case summary document

LIST all of the use cases cross-listed with the file names of actual document

## 4.2 Software Interface Description

The software interface(s)to the outside world is(are) described.

### 4.2.1 External Machine Interfaces

Interfaces to other machines (computers or devices) are described.

### 4.2.2 External System Interfaces

Interfaces to other systems, products or networks are described.

### 4.2.3 Human Interface

An overview of any human interfaces to be designed for the software is presented.

## 4.3 Sequence Diagrams

Used to model the class interactions needed for the use cases.

## 4.4 Communication Diagrams

Used to model the message passing structure of the system functions.

# 5.0 Behavioral Model and Description

A description of the behavior of the software is presented.

## 5.1 Description for Software Behavior

A detailed description of major events and states is presented in this section.

### 5.1.1 Events

A listing of events (control, items) that will cause behavioral change within the system is presented.

### 5.1.2 States

A listing of states (modes of behavior) that will result as a consequence of events is presented.

## 5.2 State Transition Diagrams

Depict the manner in which the software reacts to external events.

# 6.0 Restrictions, Limitations, and Constraints

Special issues which impact the specification, design, or implementation of the software are noted here.

# 7.0 Validation Criteria

Our validation process has been planned to ensure nothing is added to our software that doesn’t work. Before we begin development for a requirement the first thing we will do is plan unit tests that the addition must pass before integration. After the developer completes a software addition that passes all the unit tests the team lead will give the addition a technical review to suggest changes and ensure that the addition is readable, understandable and that it can be easily edited. Before each integration developers who were not involved with the addition will plan and perform black box integration tests to make sure additions work with the completed parts of the project. After the project is considered complete we will conduct validation tests within the team but also have people outside Cosmosys use our software to make sure the software meets the requirements and solves our customer’s problems.

## 7.1 Classes of Tests

* **Unit Testing**Before each addition we will plan unit tests that must be passed before an addition is considered for integration. During development developers will also unit test as they go to be more time efficient.
* **Integration Testing**Before integration we will review additions for readability, and make sure additions are factorable. After review we will do integration tests on additions before merging them with the software.
  + **Top-Down Approach**We plan to use a top-down approach in development. We will develop the base parts of the software first and each developer will work on a branch of the project with each addition going through integration testing.
  + **Regression**With each new addition we will retest the entire project at key points to make sure new additions do not cause problems higher up in the project.
  + **White-Box**During white box testing we will make sure every part of our program is tested based on its logical structure. These tests will be planned after development because they are based on the internal structure of the software and will be done before validation testing***.***
* **Validation Testing**After the project is functionally complete we will test it to validate that it meets our business requirements.
  + **Configuration Review**In this step we will review the software and all associated design documentation for consistency and completion. We will also review to see if all requirements have been met.
  + **Customer Acceptance Testing**Before project completion, if time permits, we will informally present the software to the customer to get a confirmation that our software meets the requirements.
  + **Black-Box Testing**After completion we will do internal black box testing within our team with members testing parts they haven’t worked on without looking at the code. After that we will have people outside the team use our software to make sure functional requirements are met.
* **Performance Testing**  
  We have set performance bounds (described in section 7.3) for speed of execution that we will test for after the project is complete.
  + **Refactoring**If our software fails performance tests we will refactor it to minimize time issues before deployment.
* **Deployment Testing**   
  Our software needs to work for our customer so we will test it on computers at University of Michigan -Dearborn to make sure it can work during our presentation and grading.

## 7.2 Expected Software Response

The expected results from testing are specified.

## 7.3 Performance Bounds

To set performance standards we’ve come up with two speed limitations for the types of user actions in the software. Every user action that doesn’t display sales or fuel history or data for multiple entities must be complete within 5 seconds. Every user input that involves displaying sales or fuel data over a day or week or displaying information about multiple entities must complete in 10 seconds.

# 8.0 Appendices

Presents information that supplements the Requirements Specification

## 8.1 System Traceability Matrix

A matrix that traces stated software requirements back to the system specification.

## 8.2 Product Strategies

If the specification is developed for a product, a description of relevant product strategy is presented here.

## 8.3 Analysis Metrics to be Used

The following is a description of the goal of the metrics used during this project, definitions of the various metrics used, and their applicability to the different aspects of the project.

### 8.3.1 Goal of the Metrics

Analyze the *Ice Cream Manager* architecture for the purpose of evaluating architectural components with respect to the ability to make *Ice Cream Manager* more extensible from the viewpoint of the software engineers performing the work in the context of product enhancement until deployment to the customer.

Towards that goal, the metrics that are collected will be used to answer the following questions:

1. This is the test. This is the test. This is the test. This is the test. This is the test. This is the test. This is the test.

### 8.3.2 Requirements Model Metrics

The quality of the Requirements Model for this project will be determined using two different metrics; specificity and completeness.

#### Specificity

Specificity is the measure of the precision of language and lack of ambiguity in the Requirements Model. Measurement of specificity occurs through the interpretation of requirements by reviewers. The frequency in which reviewers agree regarding how a requirement is interpreted improves the specificity score. The specificity score is calculated using the following equation:

Where is the specificity score, is the number of requirements for which all the reviewers had identical interpretations, and is the total number of functional and non-functional requirements. When interpreting the specificity score the closer it is to the more specific—or less ambiguous—the requirements.

#### Completeness

Completeness is the measure of how complete the Requirements Model is. Measurement of completeness occurs through examining the ratio between the number of functional requirements and the number of the inputs and states in the model. The completeness score is calculated using the following equation:

Where is the completeness score, is the number of unique functional requirements, is the number of inputs defined or implied by the specification, and is the number of states defined by the specification. When interpreting the completeness score the closer it is to the higher the percentage of necessary functions that have been specified.

### 8.3.3 Object-Oriented Model Metrics

The quality of the Object-Oriented Model for this project will be determined using two different metrics; responsibility and linking.

#### Responsibility

Responsibility is the measure of how much a single class is responsible for performing. Measurement of the responsibility score for a single class occurs though examining the number of methods within or inherited by the class, and the number of attributes within or inherited by the class. The responsibility score is calculated using the following equations:

Where is the overall responsibility score, is the method responsibility score, is the attribute responsibility score, is the number of methods within the class, is the number of methods inherited by the class, is the number of attributes within the class, and is the number of attributes inherited by the class. Properties—attribute getters/setters in *C#*—are not included when counting any methods or attributes for this calculation. When interpreting the method or attribute responsibility scores any values higher than is likely indicative of a need to spread the responsibilities to a new or existing class. Likewise, an overall responsibility score with a value higher than is also indicative of a class that needs its responsibilities lightened.

#### Linking

### 8.3.4 Source Code Quality Metrics

The quality of the source code for this project will be determined using two different metrics; operations and parameters.

#### Operations

#### Parameters

## 8.4 Supplementary Information (as required)

# 9.0 Software Requirements Specification Review and Signoff

Review and Signoff of the Software Requirements Specification Document.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Project Team Role | Signature | Date |
| Camille Williams | Project Manager |  |  |
| Marc King | Team Lead |  |  |
| Aly Lakhani | Developer |  |  |
| Rodney Lewis | Developer |  |  |
| Jacob Vacheresse | Developer |  |  |
| Fan Zhang | Developer |  |  |