**ESCan**

Software Requirements Specification

Version: 1.0

12/13/11

Sean Burke

Lead Software Engineer

Prepared for

Engineering Student Council

at the

University of California, Irvine

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Comments** |
| 12/13/11 | Version 1.1 | Sean Burke | The first version |
| 3/5/12 | Version 1.2 | Sean Burke | Includes Analysis following E-Week |
|  |  |  |  |
|  |  |  |  |

# Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
|  | Sean Burke | Lead Software Engineer | 12/13/11 |
|  | Sean Burke | Lead Software Engineer | 3/5/12 |

**Table of Contents**

Revision History ii

Document Approval ii

1. Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Definitions, Acronyms, and Abbreviations 1

1.4 References 1

1.5 Overview 1

2. General Description 1

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Characteristics 2

2.4 General Constraints 2

2.5 Assumptions and Dependencies 2

3. Specific Requirements 2

3.1 External Interface Requirements 3

3.1.1 User Interfaces 3

3.1.2 Hardware Interfaces 3

3.1.3 Software Interfaces 3

3.1.4 Communications Interfaces 3

3.2 Functional Requirements 3

3.2.1 <Functional Requirement or Feature #1> 3

3.2.2 <Functional Requirement or Feature #2> 3

3.3 Use Cases 3

3.3.1 Use Case #1 3

3.3.2 Use Case #2 3

3.4 Classes / Objects 3

3.4.1 <Class / Object #1> 3

3.4.2 <Class / Object #2> 3

3.5 Non-Functional Requirements 3

3.5.1 Performance 4

3.5.2 Reliability 4

3.5.3 Availability 4

3.5.4 Security 4

3.5.5 Maintainability 4

3.5.6 Portability 4

3.6 Inverse Requirements 4

3.7 Design Constraints 4

3.8 Logical Database Requirements 4

3.9 Other Requirements 4

4. Analysis Models 4

4.1 Sequence Diagrams 4

4.3 Data Flow Diagrams (DFD) 4

4.2 State-Transition Diagrams (STD) 4

5. Change Management Process 4

A. Appendices 5

A.1 Appendix 1 5

A.2 Appendix 2 5

**Introduction**

ESCan was an idea inspired by the California Polytechnic State University Engineering Student Council, where the Engineering Student Council uses a system that uses "passports" that will track the attendance of each participant. The ESCan system developed by Sean Burke at the University of California uses a similar idea, but customized for UCI. UCI's system uses barcode wristbands that participants wear that allow them to attend each event, and provide statistics for each event. At the end of the week, users who participated in the most events are entered into an opportunity drawing to win prizes.

**Goal**

The goal of ESCan is to provide statistical data on the participation of E-Week to improve the planning for upcoming E-Weeks. This is the first year in 39 years that we were able to get an accurate representation of the students who come out to E-Week. The system provide the Engineering Student Council with data such as how many people participated in each event, which majors came out to which events, which level of student and faculty were present at each event, and which time of day that had the most traffic. With this data, the Engineering Student Council will be able to plan for future events and improve details such as better start times for events, and better advertising to majors and levels of students who were lacking in participation.

**Stages**

The system had a total of 5 stages of process to go through.

1. Development

2. Testing

3. Setup

4. Training

5. Go Live

1) User Registration

2) Scanning

6. Closing/Analysis

**Development**

Sean Burke created the application entirely from scratch using original code which he single-handedly developed in 5 weeks. He began this project at the beginning of Winter Break, and spent hours developing the site which was completed at the end of Winter Break. The system consists of two parts: Software and Hardware. The software is a PHP, MySQL, CSS, Javascript, HTML website that consists of 9252 lines of code, and uses 3 barcode scanners which he found at a discount from an electronics store near his hometown. The features of the website include a user login feature where users who are registered with the system can login in and have different access to certain features, it can detect if a registered participant is an Engineer to deter non-Engineers from getting free stuff, it can detect if a user has already participated to avoid situations such as running out of food, It can detect the level and major, as well as if the person is Faculty/Staff or a Student all based on the UCInetID provided, email notification to participants, and it also provides graphs and charts of the participation in E-Week.

**Testing**

Although development took 5 weeks to complete, testing the application took from the beginning of Winter Quarter to the start of E-Week to detect every bug. There were multiple test cases such as "What happens if a barcode was already scanned?", "What if a barcode does not exist?", or "What if the UCI Directory says the person isn't an Engineer, but they in fact are an Engineer?" Each of these cases were created and tested to make sure the system performed according to planned.

**Setup**

Before the system can go live, it needs to be setup to have initial users and to initialize each wristband barcode. Sean purchased 2000 barcode wristbands from a third party and had volunteers scan each barcode to initialize the barcodes. This needed to be done in order to avoid fake barcodes during the event, otherwise any barcode could be scanned, by passing the system. In addition, each event during E-Week needed to be created through the website so that data can be associated with each event.

**Training**

In order to have a successful performance of the system, there also needs to be trained users that know how to use the system. Sean set up a ESCan Training meeting where he trained each volunteer to learn the system. After the volunteer was trained, he changed their status to Certified ESCanner which would allow that user to now login and scan each participant during E-Week.

**Go Live**

There are two stages of the process once the system is up and running. The first stage is registration and the second is scanning. In the registration stage, a Certified ESCanner will ask for a participant's UCInetID, and the system will register the participant to a wristband along with their major and level. Once a participant has a wristband, they can now scan in at each event. With each scan, the Engineering Student Council can see the collective data of each user.

**Closing/Analysis**

After the conclusion of E-Week, the Engineering Student Council can analyze the data to plan for next year's events. From the data we learned some new information that we never had in previous years. We learned that there needs to be better advertising to the Freshman Class, because of the 1000 participants, only 21 participants were registered freshman. We also learned that we receive a lot of participation from grad students which is great because we never even thought to advertise the events to grad students. We also learned that we could end the Career Fair at 2:00PM instead of 3:00PM because the participation dies down at around 2:00PM. All these analyses can be concluded with the implementation of ESCan.

# 1. Introduction

In the Winter of each year, students gather to celebrate E-Week at UCI. During this week, many events are held, and it is important to track which students are attending which events for a whole host of reasons. The E-Scan system will be able to keep statistics for each event in order to provide better planning for the following years. This document describes the system requirements for the system.

## 1.1 Purpose

The purpose of the system is to provide statistics on the students attending each event. These statistics will help the Engineering Student Council plan for future events and be able to have an accurate number for the amount of people, which attend each event. This system will also provide a convenient way for crowd control. Students will also be able to win prizes provided by the Engineering Student Council based on attendance to each event.

## 1.2 Scope

This system was inspired by the Engineering Student Council at the California Polytechnic State University, San Luis Obispo. This will be the second year that E-scan has been implemented, however this will be the first year that the software is developed at UCI. The previous system was developed by the Engineering Student Council at Cal Poly. This system shall be completed before the beginning of February to ensure proper testing is in place. The budget will be determined at a later time.

## 1.3 Definitions, Acronyms, and Abbreviations

This subsection should 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 appendixes in the SRS or by reference to other documents.

Cal Poly - California Polytechnic State University, San Luis Obispo

ESC – Engineering Student Council

UCI – University of California, Irvine

ESO – Engineering Student Organization

## 1.4 References

This subsection should:

(1) Provide a complete list of all documents referenced elsewhere in the SRS, or in a separate, specified document.

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

(3) Specify the sources from which the references can be obtained.

This information may be provided by reference to an appendix or to another document.

## 1.5 Overview

This subsection should:

(1) Describe what the rest of the SRS contains

(2) Explain how the SRS is organized.

# 2. General Description

## 2.1 Product Perspective

The system may require multiple scanners, and should be accessible from any computer with Internet access. Users of the system will include Participants, Volunteers, Administrators, and Developers. Each user will be able to log into the system website, and perform different tasks described in section 3.

## 2.2 Product Functions

This

## 2.3 User Characteristics

This subsection of the SRS should describe those general characteristics of the eventual users of the product that will affect the specific requirements. (See the IEEE Guide to SRS for more details).

## 2.4 General Constraints

This subsection of the SRS should provide a general description of any other items that will

limit the developer’s options for designing the system. (See the IEEE Guide to SRS for a partial list of possible general constraints).

## 2.5 Assumptions and Dependencies

This subsection of the SRS should 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 will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

# 3. Specific Requirements

The specific requirements for the E-Scan system are divided into the essential

and extension categories. The essential requirements are necessary to fully complete

the system and will have a high priority. The extension requirements would add

additional functionality to the sub-system, but are not necessary to complete it and will

have a low priority.

## 3.1 Essential Requirements

3.1.1 Functional Requirements

3.1.1.1 Login

3.1.1.1.1 E-Scan shall allow login for each user described in the user section

3.1.1.1.2 E-Scan shall

3.1.1.2 Scan

3.1.1.2.1 E-Scan shall allow only Volunteers, Admins and Developers to scan people into the system.

3.1.1.2.2 E-Scan shall not allow participants to scan in under any circumstances.

3.1.1.2.3 E-Scan shall display the event being scanned which is obvious to the user

3.1.1.2.4 E-Scan shall display a confirmation after scanning

3.1.1.2.5 E-Scan shall display all scans instantaneoulsy for the event in an easy to scroll list.

3.1.1.3 Display Statistics

3.1.1.3.1 E-Scan shall graph the total number of students registered to attend E-Week in a bar graph

3.1.1.3.2 E-Scan shall graph the total number of students scanned into each event in a bar graph

3.1.1.3.3 E-Scan shall graph the total number of each major attending all events in a pie graph

3.1.1.3.4 E-Scan shall calculate the total number of students registed vs the total number of students attended

3.1.1.3.5 E-scan

3.1.1.4 Determine Prize Winner

3.1.1.5 Display Events

3.1.1.5.1 E-Scan shall display a list of events for E-Week

3.1.1.6 Register

3.1.1.6.1 E-Scan shall allow users to register with their UCInetID.

3.1.1.6.2 E-Scan shall determine whether the user is an engineer during registration.

3.1.1.6.3 E-Scan shall allow users to request whether to receive emails during registration.

3.1.1.6.4 E-Scan shall require users to enter a UCInetID when registering.

3.1.1.6.5 E-Scan shall be able to find all information decribed in the registration process.

3.1.1.6.6 E-Scan shall send an email confirmation to the user after registration

3.1.1.6.7 E-Scan shall encrypt passwords before storing them

This will be the largest and most important section of the SRS. The customer requirements will be embodied within Section 2, but this section will give the D-requirements that are used to guide the project’s software design, implementation, and testing.

Each requirement in this section should be:

* Correct
* Traceable (both forward and backward to prior/future artifacts)
* Unambiguous
* Verifiable (i.e., testable)
* Prioritized (with respect to importance and/or stability)
* Complete
* Consistent
* Uniquely identifiable (usually via numbering like 3.4.5.6)

Attention should be paid to the carefuly organize the requirements presented in this section so that they may easily accessed and understood. Furthermore, this SRS is not the software design document, therefore one should avoid the tendency to over-constrain (and therefore design) the software project within this SRS.

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

### 3.1.2 Hardware Interfaces

### 3.1.3 Software Interfaces

### 3.1.4 Communications Interfaces

## 3.2 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

### 3.2.1 <Functional Requirement or Feature #1>

3.2.1.1 Introduction

3.2.1.2 Inputs

3.2.1.3 Processing

3.2.1.4 Outputs

3.2.1.5 Error Handling

### 3.2.2 <Functional Requirement or Feature #2>

…

## 3.3 Use Cases

### 3.3.1 Use Case #1

### 3.3.2 Use Case #2

…

## 3.4 Classes / Objects

### 3.4.1 <Class / Object #1>

3.4.1.1 Attributes

3.4.1.2 Functions

<Reference to functional requirements and/or use cases>

### 3.4.2 <Class / Object #2>

…

## 3.5 Non-Functional Requirements

Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transaction shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc).

### 3.5.1 Performance

### 3.5.2 Reliability

### 3.5.3 Availability

### 3.5.4 Security

### 3.5.5 Maintainability

### 3.5.6 Portability

## 3.6 Inverse Requirements

State any \*useful\* inverse requirements.

## 3.7 Design Constraints

Specify design constrains imposed by other standards, company policies, hardware limitation, etc. that will impact this software project.

## 3.8 Logical Database Requirements

Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.

## 3.9 Other Requirements

Catchall section for any additional requirements.

# 4. Analysis Models

List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable the SRS’s requirements.

## 4.1 Sequence Diagrams

## 4.3 Data Flow Diagrams (DFD)

## 4.2 State-Transition Diagrams (STD)

# 5. Change Management Process

Identify and describe the process that will be used to update the SRS, as needed, when project scope or requirements change. Who can submit changes and by what means, and how will these changes be approved.

# A. Appendices

Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS’s overall set of requirements.

Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.

## A.1 Appendix 1

## A.2 Appendix 2