**CS360**

**Software Requirements Specification (SRS) Document**

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).

**CS360**

**Habegger, Parker, Rasler**

**Egg Alert and Real-Time Logistics (EARL)**

**Software Requirements Specification**

**Document**

**Version: (n)** **Date: (mm/dd/yyyy)**

**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) outlines the purpose of this document in relation to the product specified: The Egg Flow Communicator (EFC).

## 1.1 Purpose

The purpose of this Software Requirements Specifications document is to list and detail the requirements inherent in the construction and maintenance of the Egg Flow Communicator. It is intended for the use of the client to verify that all required specifications for the EFC have been listed and considered. Further, the SRS will be used by the development team to ensure that all required design parameters are incorporated into the final product.

## 1.2 Scope

Software product to be produced: Software aspect of the Egg Flow Communicator

Creation of a system that automates the process of determining when and where a chicken egg flow problem (egg jam) occurs on a system of conveyors through the chicken egg packaging process. Mechanical units will be installed along separate conveyors to track the flow of eggs down that specific conveyor; these units will report to a software program designed to determine if the flow is normal or abnormal. In the case of abnormal flow, the system will alert the user in real-time as to which specific line the problem has occurred on. In a typical poultry operation, thousands of feet of conveyor lines would need to be searched manually to locate a jam. This system would minimize the searching, thus minimizing the labor cost needed to fix the problem. The system also eliminates unnecessary loss in performance by alerting users even when the packaging system is not in use.

## 1.3 Definitions, Acronyms, and Abbreviations.

EARL: Egg Alert and Real-time Logistics

UART: Universal Asynchronous Receiver/Transmitter

BS2P40: BASIC stamp 2p 40-pin microcontroller module

CAT-5: Serial twisted pair category 5 RJ45 style cabling

## 1.4 References

Voice of Customer Document

Functional Requirments Document

Web Link: <http://cmapspublic.ihmc.us/rid=1K1K884SC-X1XFD0-2TWB/RaslerSoftEngCmap.cmap>

## 1.5 Overview

This document is segmented in such a way that the scope of the primary sections 1-2 is most suitable for those not involved in the construction of the Egg Flow Communicator. The primary section 3 is most suitable for those involved in the construction of the software.

# 2. The Overall Description

## 2.1 Product Perspective

# This product is being created in an environment with the following already established requisites: A production line is already established with individual conveyors fed from lines of chicken coops into primary lines that feed into the packaging area, and a windows PC with touch screen exists at the user-operated packaging area.

A preexisting product already established at the intended location is best described as a mechanical sorting device that uses optical recognition and software to remove specific sized eggs from the conveyor. The mechanized solution being elaborated upon in this document will work, and be implemented alongside this preexisting system.

### 2.1.1 System Interfaces

This software system is to be operated on a PC which also operates an existing system that detects egg size and removes eggs too large for packing. The two systems’ GUIs will share the existing touch screen monitor, but otherwise will not interact; menu options to access alarms indicated by the EFC will be incorporated into the GUI.

Likewise, the sensor system must be integrated into the existing conveyor system. These will be installed according to specifications made by the client.

### 

### 2.1.2 Interfaces

A preexisting interface exists for the mechanized egg sorting device. The Egg Flow Communicator will necessarily be integrated into this preexisting interface. The interface includes a GUI developed specific for a preexisting touch screen for user input and display. The touch screen input will drive the user adjustable parameters, alert toggles, and so on.

The GUI aspect specific for this application should allow easily adjustable parameters, and should provide a viewable

### 

### 2.1.3 Hardware Interfaces

Mechanical counter devices should be installed on the conveyor in such a way as to gauge the flow rate of that particular conveyor belt. These sensors communicate with a conveyor specific microcontroller (BS2P40), and utilizing a UART to communicate via serial lines, these serial transmissions should be converted to a USB specified transmission. This USB transmission will interface with a preexisting PC that should interpret and drive the communication of the whole system.

### 

### 2.1.4 Software Interfaces

2.1.4.1 Software Interface of BS2P40:

The Sponsor specific microcontrollers require PBASIC to communicate serially with the UARTs.

2.1.4.2 Software Interface of the primary program:

The primary program will handle communication to the UARTs. This level of communication should be programmed for the Windows Environment. The communication data should interface with software that knows the state of each conveyor, utilizing algorithms to determine if an alert or response is necessary. This level of the software interfaces with a GUI, making alerts as necessary, and describing the status of the conveyors. The preexisting touch screen should offer the user the ability to modify internal settings of the software, specifically including the ability to modify sensitivity settings and alert settings.

### 2.1.5 Communications Interfaces

2.1.5.1 Communication Interface for UART to primary program software:

The communication to the UARTs to the PC will be established after the purchase of the chips, the specific chip will determine the protocol necessary, and may come prepackaged with the chip. The data will be transmitted over Cat-5 serial cable.

2.1.5.2 Communication Interface from UARTs to BS2P40:

These two components will communicate at the hardware level as specified by the microcontroller and UART chosen.

2.1.5.3 Communication Interface from BS2P40 to conveyor positioned egg detector:

These two components will communicate at the hardware level as specified by the microcontroller.

### 2.1.6 Memory Constraints

This system’s memory requirements will not exceed the memory allowance of the preexisting system.

### 2.1.7 Operations

The user will interact with a GUI located at a packaging station, responding to alerts from the program by noting the location that the alert originated, and manually fixing the problem.

### 2.1.8 Site Adaptation Requirements

Site adaptations would include:

Mechanical counters installed at intervals along individual conveyor lines.

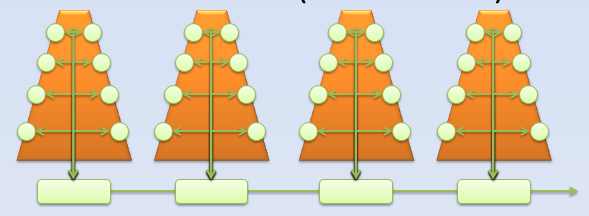
Housing boxes established to hold both the UART and the BS2P40.

Wiring installed connecting counters to housing boxes.

Serial communication lines installed from each housing box to a workstation PC.

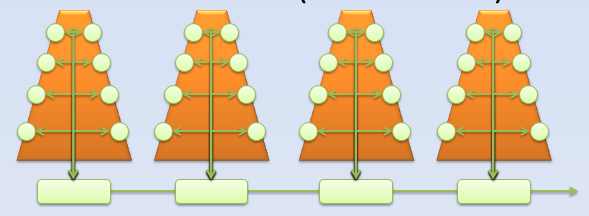
Note that, though these requirements are listed above, they are still considered in this document to be part of the overall system. They are listed above to aid in specifying a more generic system, rather than the specific instance that this document deals with.

## 2.2 Product Functions



Mechanical egg detectors (white circles) placed on conveyor lines (orange bands) detect the presence of eggs as they move down conveyor belts.

Egg Jam Here!



Alert!

However, conveyance can stop due to mechanical issues or a system overload (egg-jam). Users are often remote and unaware of the problem. When eggs stop being detected on one line, or one location (red circles) and other detectors are detecting egg flow, then there is a jam.

Because eggs flow at differing rates, the software is responsible for determining what is a jam and what is not. It is also responsible for determining error situations. For example: if a module is unresponsive and other modules are responsive, then the module needs maintenance.

The program is also responsible for intelligibly displaying the status of the system. It should also provide methods for alerting that are expressive.

## 2.3 User Characteristics

The educational level, experience, and technical expertise required by the user is no more than required by methods already established in the process.

The user will be involved in the packaging process while operating this system, so considerations involving speed of usability and economy of motion specific to the user are of import.

## 2.4 Constraints

The choice of hardware components will constrain methods used to communicate with the UARTs and BS2P40s. This is yet to be determined.

The software will be hosted on a PC utilizing the Windows Operating System.

## 2.5 Assumptions and Dependencies

The characteristic assumption of the system is that an egg jam can in fact be determined from the given amount of detectors installed.

.Net should be installed prior to the system being installed, as a prerequisite.

# 3. Specific Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| FR1 | Presentation Layer | DP1 |  |
| FR2 | Business Layer | DP2 |  |
| FR3 | Data Access Layer | DP3 |  |
| FR4 | Persistence Layer | DP4 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| FR1.1 | Shall have a GUI | DP1.1 | Windows Forms |
| FR1.2 | Shall work with UI Process Components | DP1.2 | Windows Forms |
| FR1.3 | Shall utilize the preexitsting Touch Screen | DP1.3 | Utilize Existing Touch Screen |

|  |  |  |  |
| --- | --- | --- | --- |
| FR1.1.1 | Shall have a Configuration Interface | DP1.1.1 | Form which displays configuration settings and allows for changing settings |
| FR1.1.2 | Shall have a Visual Alert | DP1.1.2 | Generate window with alert information |
| FR1.1.3 | Shall have an Audio Alert | DP1.1.3 | Generate alert sound using computer speakers |
| FR1.1.4 | Shall have a Status Screen | DP1.1.4 | Generate window with status information |

|  |  |  |  |
| --- | --- | --- | --- |
| FR2.1 | Shall run on Windows OS | DP2.1 | Utilize Existing POS |
| FR2.2 | BS2P40 | DP2.2 | Utilize the BS2P40 Microcontroller |

|  |  |  |  |
| --- | --- | --- | --- |
| FR2.1.1 | Primary Application | DP2.1.1 | .Net or Java Based Application |
| FR2.1.2 | Shall utilize a Serial Communication Driver | DP2.1.2 | C driver created to drive communication |
| FR2.1.3 | Shall utilize the USB Standard | DP2.1.3 | Software USB capable |
| FR2.1.4 | Shall utilize the RS485 Standard | DP2.1.4 | Convert communication signal to RS485 |

|  |  |  |  |
| --- | --- | --- | --- |
| FR2.1.1.1 | Shall have a Bridge to Communication Driver | DP2.1.1.1 | Java: JNI , C++:Link C driver; both utilizing windows.h |
| FR2.1.1.2 | Shall have a Hardware Polling Process | DP2.1.1.2 | .Net or Java Control Structure Implemented to Poll Modules |
| FR2.1.1.3 | Shall have a State Logic Process | DP2.1.1.3 | .Net or Java Control Structure Implemented as a State Machine |
| FR2.1.1.4 | Shall have a GUI Build/Update | DP2.1.1.4 | .Net: Windows Forms, Java:Swing |

|  |  |  |  |
| --- | --- | --- | --- |
| FR2.2.1 | Shall have a RS485 to RS242 Convertor | DP2.2.1 | Hardware Converter for RS232 to RS485 |
| FR2.2.2 | Shall utilize an UART | DP2.2.2 | Utilize provided UART |
| FR2.2.3 | Shall utilize Communication Software | DP2.2.3 | Hardware Implemented |
| FR2.2.4 | Shall have Counting Software | DP2.2.4 | PBASIC Control Structure to count eggs from hardware device |

|  |  |  |  |
| --- | --- | --- | --- |
| FR3.1 | Primary Application | DP3.1 | .Net or Java Based Application |

|  |  |  |  |
| --- | --- | --- | --- |
| FR3.1.1 | Shall utilize a File Writer | DP3.1.1 | .Net: TextFileWriter, Java:FileWriter |
| FR3.1.2 | Shall utilize a File Reader | DP3.1.2 | .Net: TextFileReader, Java:FileReader |
| FR3.1.3 | Shall have a Parser/Tokenizer | DP3.1.3 | .Net or Java based object to parse .ini files |

|  |  |  |  |
| --- | --- | --- | --- |
| FR4.1 | Primary Application | DP4.1 | .Net or Java Based Application |
| FR4.2 | BS2P40 Microcontroller | DP4.2 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| FR4.1.1 | Shall utilize Configuration Settings | DP4.1.1 | Configuration txt file |
| FR4.1.2 | Shall have Log Files | DP4.1.2 | log txt file |
| FR4.1.3 | Shall have State Values | DP4.1.3 | Configuration txt file |

|  |  |  |  |
| --- | --- | --- | --- |
| FR4.2.1 | Shall have an Address | DP4.2.1 | PBASIC Object stores address |
| FR4.2.2 | Shall have State Values | DP4.2.2 | PBASIC Objects can remember state |

## 3.1 External Interfaces

3.1.1 Primary Application Interface

3.1.1.1 Component Interfaces

The primary application should interface through RS485 connection, communicating to a UART (provided by client) by serial to communicate the current state and count of each individual module.

3.1.1.2 User Interfaces

The user should communicate through touch screen to adjust settings, control alerts, view logging information, etc..

3.1.2 UART Interface

The primary application should communicate with the UART (provided by client) acting as an intermediary to the communication line from the module (controlled by the microcontroller) to the primary application. This communication should be RS232, interfacing to a RS485 converter.

3.1.3 Microcontroller Interface

The microcontroller (provided by client) should interface with the primary application through an intermediary UART, so effectively, it should interface directly to the UART, communicating through whatever means is constrained by the provided components (TTL).

## 3.2 Functions

FR0: Egg Flow Communicator  
   FR1: Presentation Layer  
      FR1.1: Shall have a GUI  
         FR1.1.1: Shall have a Configuration Interface  
         FR1.1.2: Shall have a Visual Alert  
         FR1.1.3: Shall have an Audio Alert  
         FR1.1.4: Shall have a Status Screen  
      FR1.2: Shall work with UI Process Components  
      FR1.3: Shall utilize the preexitsting Touch Screen  
   FR2: Business Layer  
      FR2.1: Shall run on Windows OS  
         FR2.1.1: Primary Application  
            FR2.1.1.1: Shall have a Bridge to Communication Driver  
            FR2.1.1.2: Shall have a Hardware Polling Process  
            FR2.1.1.3: Shall have a State Logic Process  
            FR2.1.1.4: Shall have a GUI Build/Update  
         FR2.1.2: Shall utilize a Serial Communication Driver  
         FR2.1.3: Shall utilize the USB Standard  
         FR2.1.4: Shall utilize the RS485 Standard  
      FR2.2: BS2P40  
         FR2.2.1: Shall have a RS485 to RS242 Convertor  
         FR2.2.2: Shall utilize an UART  
         FR2.2.3: Shall utilize Communication Software  
         FR2.2.4: Shall have Counting Software  
   FR3: Data Access Layer  
      FR3.1: Primary Application  
         FR3.1.1: Shall utilize a File Writer  
         FR3.1.2: Shall utilize a File Reader  
         FR3.1.3: Shall have a Parser/Tokenizer  
   FR4: Persistence Layer  
      FR4.1: Primary Application  
         FR4.1.1: Shall utilize Configuration Settings  
         FR4.1.2: Shall have Log Files  
         FR4.1.3: Shall have State Values  
      FR4.2: BS2P40 Microcontroller  
         FR4.2.1: Shall have an Address  
         FR4.2.2: Shall have State Values

## 3.3 Performance Requirements

<!MEETING ELABORATE>

*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 Andrew

<!MEETING ELABORATE>

*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 Andrew

*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

### 3.6.1 Reliability

The Egg Flow Communicator should be able to identify a jam situation within 8 minutes from the initiation of stoppage at least 95% of the time. False positives (e.g. a particularly long break in egg production interpreted as a stoppage) should be limited to less than one occurrence per operation.

### 3.6.2 Availability

It is necessary for the Egg Flow Communicator to be available on demand, as the encapsulating conveyor system is run on a variable schedule. It should be able to operate daily for a minimum of 18 hours continuously. In the event of a system failure, the EFC should be able to be recovered within 5 minutes.

### 3.6.3 Security

The function of the Egg Flow Communicator is self-contained. The only interaction between the EFC and other programs is that its GUI is displayed concurrently on the same monitor as the GUI for the egg-sizing robot that is operated by the PC. The only trading of data occurs during the writing of log files. Should the EFC crash, it will not affect the operation of other programs. Further, should there be a malfunction or failure of a system module, the EFC will be able to report the occurrence and identify the bad microcontroller or UART.

### 3.6.4 Maintainability

The function of the EFC is divided into the following modules:

* GUI: Touch-screen interface for locating jams, disabling alarms, and viewing logs.
* Stamp programming: software for the BASIC microcontrollers.
* Jam logic: determination of a jam or other error from the collected input signals.
* Alert system: initiates visual and audial warnings to the user interfaces.
* Logging system: records time and locations of jams.

### 3.6.5 Portability

Portability is not a strong consideration for the EFC. The software is being tailored to integrate with a legacy system using components that the client already owns or is familiar with. That said, certain portions of the software will be portable, as designated below:

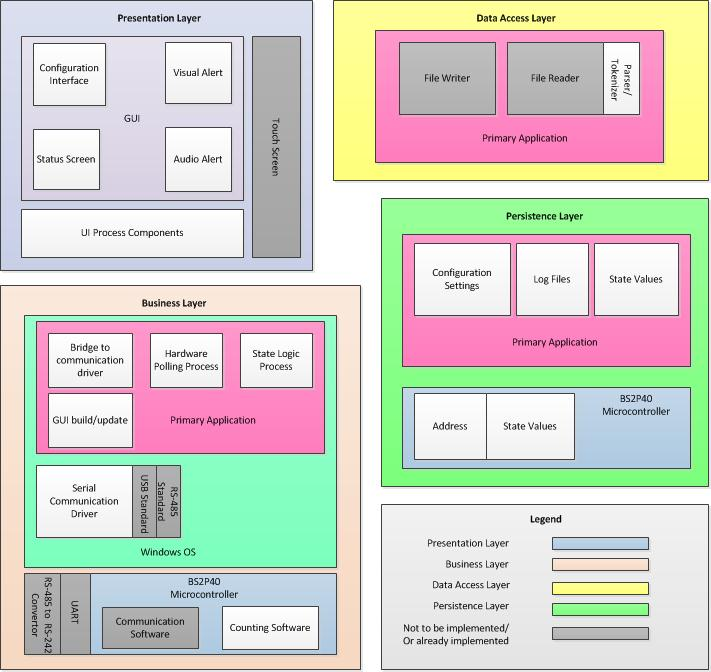
* Software for the UARTs and BASIC stamp units will be coded in Parallax BASIC. This code is specific to the chosen equipment and may require retooling if these components should become unavailable.
* The main program (signal interpretation, jam determination and location, alerts, and user interface) will be coded in .NET. While the intended platform is a Windows-based PC, this part could potentially be ported to other operating systems. We will not explore this possibility in the scope of this project.

### 3.6.6 Usability

The Egg Flow Communicator is most likely to be used by nontechnical personnel. As such, the user interfaces will stress ease of use. The GUI should be minimalistic and menu-driven, featuring components that are oversized and easy to read.

## 3.7 Organizing the Specific Requirements

3.7.1 Requirements Organized by Apllication Architecture



Each logical section will create a heirarchial node to nest the requirements as such:

FR0: Egg Flow Communicator  
   FR1: Presentation Layer  
      FR1.1: Shall have a GUI  
         FR1.1.1: Shall have a Configuration Interface  
         FR1.1.2: Shall have a Visual Alert  
         FR1.1.3: Shall have an Audio Alert  
         FR1.1.4: Shall have a Status Screen  
      FR1.2: Shall work with UI Process Components  
      FR1.3: Shall utilize the preexitsting Touch Screen  
   FR2: Business Layer  
      FR2.1: Shall run on Windows OS  
         FR2.1.1: Primary Application  
            FR2.1.1.1: Shall have a Bridge to Communication Driver  
            FR2.1.1.2: Shall have a Hardware Polling Process  
            FR2.1.1.3: Shall have a State Logic Process  
            FR2.1.1.4: Shall have a GUI Build/Update  
         FR2.1.2: Shall utilize a Serial Communication Driver  
         FR2.1.3: Shall utilize the USB Standard  
         FR2.1.4: Shall utilize the RS485 Standard  
      FR2.2: BS2P40  
         FR2.2.1: Shall have a RS485 to RS242 Convertor  
         FR2.2.2: Shall utilize an UART  
         FR2.2.3: Shall utilize Communication Software  
         FR2.2.4: Shall have Counting Software  
   FR3: Data Access Layer  
      FR3.1: Primary Application  
         FR3.1.1: Shall utilize a File Writer  
         FR3.1.2: Shall utilize a File Reader  
         FR3.1.3: Shall have a Parser/Tokenizer  
   FR4: Persistence Layer  
      FR4.1: Primary Application  
         FR4.1.1: Shall utilize Configuration Settings  
         FR4.1.2: Shall have Log Files  
         FR4.1.3: Shall have State Values  
      FR4.2: BS2P40 Microcontroller  
         FR4.2.1: Shall have an Address  
         FR4.2.2: Shall have State Values

## 3.8 Additional Comments

The Axiomatic Design Process was used to map individual Functional Requirements to Design Parameters. These mappings were used to isolate instances of redundancies. The Design Matrix is shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | DP1: | DP2: | DP3: | DP4: |
| FR1: Presentation Layer | X |  |  |  |
| FR2: Business Layer |  | X |  |  |
| FR3: Data Access Layer |  |  | O |  |
| FR4: Persistence Layer |  |  |  | O |

|  |  |  |  |
| --- | --- | --- | --- |
|  | DP1.1: Windows Forms | DP1.2: Windows Forms | DP1.3: Utilize Existing Touch Screen |
| FR1.1: Shall have a GUI | X | X | X |
| FR1.2: Shall work with UI Process Components | O | X |  |
| FR1.3: Shall utilize the preexitsting Touch Screen |  |  | X |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | DP1.1.1: Form which displays configuration settings and allows for changing settings | DP1.1.2: Generate window with alert information | DP1.1.3: Generate alert sound using computer speakers | DP1.1.4: Generate window with status information |
| FR1.1.1: Shall have a Configuration Interface | X |  |  |  |
| FR1.1.2: Shall have a Visual Alert |  | X |  | O |
| FR1.1.3: Shall have an Audio Alert | X | X | X | O |
| FR1.1.4: Shall have a Status Screen |  |  |  | X |

|  |  |  |
| --- | --- | --- |
|  | DP2.1: Utilize Existing POS | DP2.2: Utilize the BS2P40 Microcontroller |
| FR2.1: Shall run on Windows OS | X |  |
| FR2.2: BS2P40 |  | X |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | DP2.1.1: .Net or Java Based Application | DP2.1.2: C driver created to drive communication | DP2.1.3: Software USB capable | DP2.1.4: Convert communication signal to RS485 |
| FR2.1.1: Primary Application | X | X | X |  |
| FR2.1.2: Shall utilize a Serial Communication Driver |  | X | X | X |
| FR2.1.3: Shall utilize the USB Standard |  |  | X |  |
| FR2.1.4: Shall utilize the RS485 Standard |  |  |  | X |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | DP2.1.1.1: Java: JNI , C++:Link C driver; both utilizing windows.h | DP2.1.1.2: .Net or Java Control Structure Implemented to Poll Modules | DP2.1.1.3: .Net or Java Control Structure Implemented as a State Machine | DP2.1.1.4: .Net: Windows Forms, Java:Swing |
| FR2.1.1.1: Shall have a Bridge to Communication Driver | X |  |  |  |
| FR2.1.1.2: Shall have a Hardware Polling Process | X | X | O |  |
| FR2.1.1.3: Shall have a State Logic Process |  | O | X |  |
| FR2.1.1.4: Shall have a GUI Build/Update |  |  |  | X |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | DP2.2.1: Hardware Converter for RS232 to RS485 | DP2.2.2: Utilize provided UART | DP2.2.3: Hardware Implemented | DP2.2.4: PBASIC Control Structure to count eggs from hardware device |
| FR2.2.1: Shall have a RS485 to RS242 Convertor | X |  |  |  |
| FR2.2.2: Shall utilize an UART |  | X |  |  |
| FR2.2.3: Shall utilize Communication Software |  |  | X |  |
| FR2.2.4: Shall have Counting Software |  |  |  | X |

|  |  |
| --- | --- |
|  | DP3.1: .Net or Java Based Application |
| FR3.1: Primary Application | X |

|  |  |  |  |
| --- | --- | --- | --- |
|  | DP3.1.1: .Net: TextFileWriter, Java:FileWriter | DP3.1.2: .Net: TextFileReader, Java:FileReader | DP3.1.3: .Net or Java based object to parse .ini files |
| FR3.1.1: Shall utilize a File Writer | X |  |  |
| FR3.1.2: Shall utilize a File Reader |  | X |  |
| FR3.1.3: Shall have a Parser/Tokenizer |  |  | X |

|  |  |  |
| --- | --- | --- |
|  | DP4.1: .Net or Java Based Application | DP4.2: |
| FR4.1: Primary Application | X |  |
| FR4.2: BS2P40 Microcontroller |  | O |

|  |  |  |  |
| --- | --- | --- | --- |
|  | DP4.1.1: Configuration txt file | DP4.1.2: log txt file | DP4.1.3: Configuration txt file |
| FR4.1.1: Shall utilize Configuration Settings | X | O | O |
| FR4.1.2: Shall have Log Files | O | X | O |
| FR4.1.3: Shall have State Values | O | O | X |

|  |  |  |
| --- | --- | --- |
|  | DP4.2.1: PBASIC Object stores address | DP4.2.2: PBASIC Objects can remember state |
| FR4.2.1: Shall have an Address | X | O |
| FR4.2.2: Shall have State Values | O | X |

# Change Management Process

Simple changes can effectively be considered using the Axiomatic Design Tool, the residual effects observed from the streamlined creation of design views from the tool. Any changes to requirements should be formilized in a new SRS, and a team consensus should be sought before any changes are officially negotiated. A new Voice of Customer document should also be created for quality assurance and reference.

# Document Approvals

|  |  |  |
| --- | --- | --- |
| *Name* | *Signature* | *Date* |
| Matthew Rasler |  | 12/5/2011 |
| Mark Parker |  |  |
| Andrew Habegger |  |  |

# Supporting Information

# 6.1. Product Overview

Creation of a system that mechanizes the process of determining when and where a chicken egg flow problem (egg jam) occurs on a system of conveyors through the chicken egg packaging process. Mechanical units will be installed along separate conveyors to track the flow of eggs down that specific conveyor, these units will report to a software program designed to determine if the flow is normal or abnormal. In the case of abnormal flow, the system should alert the user in real-time as to which specific line the problem has occurred on. In actual use, thousands of feet of conveyor lines would need to searched manually in the instance of a jam, this system would minimize the searching, thus minimizing the labor needed to fix the problem. Also the system eliminates unnecessary loss in performance, by alerting a user even when the packaging system is not in use.