**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 Flow Communicator**

**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 Jam Communicator (EJC).

## 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 Jam Communicator. It is intended for the use of the client to verify that all required specifications for the EJC 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 Jam Communicator

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 cost 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.

## 1.3 Definitions, Acronyms, and Abbreviations.

EJC: Egg Jam Communicator

UART: Universal Asynchronous Reciever/Transmitter

BS2P40: BASIC stamp 2p 40-pin microcontroller module

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

## 1.4 References

None.

## 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 Jam 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 along side this preexisiting system.

### 2.1.1 System Interfaces

This software system must interface with an existing system that detects egg size and removes eggs that are too large for packing. The two systems will share theexisting touch screen GUI; menu options to access alarms indicated by the EJC 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 Jam Communicator will necessarily be integreated 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 commuciation 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 choosen.

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 systems 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 specifically 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 isn’t. 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.

## 2.6 Apportioning of Requirements.

Not applicable.

# 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

 **3.2.1. Presentation Layer**

 3.2.1.1 Display current state of conveyors

 3.2.1.2 Alert when state is inappropriate

 3.2.1.3 User interface adjustable parameters for sensitivity settings, etc.

 3.2.1.4 Operate on Windows platform

 3.2.1.5 Integrate touch screen capabilities

 **3.2.2. Communication Layer**

 3.2.2.1 Set up software on components for communication

 3.2.2.2 Integrate serial communication into Application

 **3.2.3 Business Layer**

 3.2.3.1 GUI driver for presentation layer

` 3.2.3.2 Communicate between persistence layer and presentation layer for settings, state, etc.

3.2.3.3 Bridge between high level software and low level communication software

 3.2.3.4 Main driving software application

 **3.2.4. Persistence Layer**

 3.2.4.1 Log files for debugging

 3.2.4.2 Settings files

 **3.2.5. Hardware Layer**

 3.2.5.1 Install hardware counters

 3.2.5.2 Install communication lines

 3.2.5.3 Install communication controllers

 3.2.5.4 Integrate into already built workstation

## 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

<!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

*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

*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

*Specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart. This is somewhat related to reliability. Some systems run only infrequently on-demand (like MS Word). Some systems have to run 24/7 (like an e-commerce web site). The required availability will greatly impact the design. What are the requirements for system recovery from a failure? “The system shall allow users to restart the application after failure with the loss of at most 12 characters of input”.*

### 3.6.3 Security

*Specify the factors that would protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to:*

1. *Utilize certain cryptographic techniques*
2. *Keep specific log or history data sets*
3. *Assign certain functions to different modules*
4. *Restrict communications between some areas of the program*
5. *Check data integrity for critical variables*

### 3.6.4 Maintainability

*Specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices. If someone else will maintain the system*

### 3.6.5 Portability

*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

*Some systems provide different sets of functions to different classes of users.*

### 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.*