Test Stand Diagnostics

Software Requirements Specification

1.2.0

4/06/2016

Aaron Monahan

Steven Draugel

John Parreno

Prepared for

Sebastian van Delden

CSCI 462 Software Engineering Practicum

# **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Comments** |
| 4/6/16 | Start of document | Aaron | Initialize layout and insert generalized information |
| 4/12/16 | Scope | Aaron |  |
| 4/22/16 | Analysis Models and FR | John |  |
| 4/22/16 | Requirements, Constraints, and Dependencies | Steven |  |
| 4/29/16 | Final Draft | Aaron, John, Steven | Finished all of the unfinished portions of the document and submitted to the professor |

# **Document Approval**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
|  | <Your Name> | Lead Software Eng. |  |
|  |  |  |  |
|  |  |  |  |

**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.................................................................................................................... 2

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.................................................................................................. 4

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................................................................................................................. 5

4.3 Data Flow Diagrams (DFD)...................................................................................................... 5

4.2 State-Transition Diagrams (STD)............................................................................................. 5

5. Change Management Process................................................................................................... 5

A. Appendices................................................................................................................................. 5

A.1 Appendix 1................................................................................................................................ 5

A.2 Appendix 2................................................................................................................................ 5

# **1. Introduction**

# **1.1 Purpose**

The purpose of this Software Requirement Specification is to inform the user on the essential characteristics of the software. This document will talk about the progression of the project, the current state and the future of the software. Our project partners Bosch are the intended audience of this and they will be able to reference this specification for any information about the software. The information in this document contains a detailed description of the software. It will be a reference for all engineers and future developers that work on the software. Topics covered include a walkthrough of the general layout, software structure diagrams, use cases, functionality and requirements.

## **1.2 Scope**

The scope of this software is for a specific industrial system. The Test Stand Diagnostics is a system monitoring tool. The software will be implemented on an industrial manufacturing facility, which is fully automated (robotic), and will diagnosing issues that occur in the final phase of the part creation. The sole use of this tool will be for the project partners, Bosch, to use on their testing systems.

This piece of software was designed to 1) Track the first pass percentage of the Anti-Lock Braking System (ABS) as they are tested 2) Notify test engineers via email when the average percentage drops below a certain threshold and 3) Once a part has been tested and shown failure, the software will diagnose the problem by parsing the data from the test sensors and display the solution to the engineer.

Bosch will benefit from this software in many ways. The first objective was to show the average first pass percentage on the testing system. The test engineers will benefit because they will be able to instantly see the status of the system displayed on their screen and determine if they need to further diagnose the system manually. Upon further investigation of a problem, they may choose to resume the tests after the problem has been resolved. The second requirement was to notify the test engineers when the system has fallen below a certain threshold of the average pass percentage. After the system has fallen below 94% of the parts passing on the first test, an email will be sent to the engineers to notify them of the status of the system. Finally, after failures occur, this tool will be able to evaluate data taken from the sensors on the testing machine. Depending on what type of ABS is currently being tested, the software will react and determine what testing point failed and display a log of the part name, reason for failure and proposed solution.

**1.3 Definitions, Acronyms, and Abbreviations**

*ABS* - Anti-Lock Braking System

*ESP* - Electronic Stability Control

*TCS* - Traction Control System

TSD - Test Stand Diagnostics

*Git* - is a version control system that is widely used for software development and other version control tasks.

*SRS* - Software Requirement Specification

*Python* - a high-level general purpose programming language

*Scrum* - Scrum is an agile software development model based on multiple small teams working in an intensive and interdependent manner.

*UML* - Unified Modeling Language

## **1.4 References**

There are no external references that this document required

## **1.5 Overview**

The rest of the this document is a detailed description of the size of the project, the constraints, requirements and some implementation. There are many different perspectives one must consider when starting a software project, we will discuss some considerations regarding the scope of the project. Finally there will be documentation on diagrams and flowcharts of the software. The diagrams are useful for understanding how each class and method interacts with each other.

**2. General Description**

**2.1 Product Perspective**

At Bosch, certain parts of the assembly line have software installed that displays statuses or general information, so the Test Stand Diagnostics is by no means a new concept. The employees at Bosch are mainly in charge of maintaining maximum uptime on the line. Workers either load parts, check parts or monitor assembly processes. Since a large percentage of the assembly is done by robots, it allows each stage to collect data and save them on a database. Certain stages of the line have computers which pull this data and display the status of the system. Although there may be other status indicators that Bosch has implemented in their manufacturing plant, the TSD has the unique functionality of being able to diagnose issues and display them to the test engineers.

## **2.2 Product Functions**

## **2.3 User Characteristics**

The users of this software will be Mechanical Engineers with limited to no knowledge of programming and as such will not have the ability to troubleshoot, maintain, or extend the source code. They will however have a higher than average ability to navigate a software system and will therefore only need minimal guidance as to the use of this software.

## **2.4 General Constraints**

The only constraints are working within Bosch’s security guidelines to which:

* No data may be passed off site
* No access to the internal database is allowed
* All other security limitations will be governed by the site’s IT department

## **2.5 Assumptions and Dependencies**

This software assumes that the following environment will be present at run time:

* Windows Operating System 7+
* Microsoft Office 2010 with Outlook
* Python 2.7 or higher

The dependencies are bundled with the software and do not require download or specific install.

# **3. Specific Requirements**

## **3.1 External Interface Requirements**

### **3.1.1 User Interfaces**

No external user interface requirements

### **3.1.2 Hardware Interfaces**

No external hardware interface requirements

### **3.1.3 Software Interfaces**

Our software does not directly interact with any interfaces. However it is worth mentioning that Bosch uses a piece of software which interacts with their database. This software takes the data coming from each step in the assembly process and parses it into understandable documents. This software takes the same raw data from the machines that the TSD uses to parse its information.

### **3.1.4 Communications Interfaces**

No external communications interface Requirements

**3.2 Functional Requirements**

Functional requirements of our software include things that Bosch decided was the most important functionality the software needed to help their test engineers. These requirements can be anything from interface design to specific back-end functions.

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

3.2.1.1 Introduction

The TSD tool is meant to help the test engineers not only monitor the system’s performance in real time but to also allow them to quickly trouble shoot issues with the testing equipment.

3.2.1.2 Inputs

The main inputs are:

* .rdy files:
  + text files that contain testing information output by the testing machines
* Excel config file called “test” used in gathering test point information like:
  + System area
  + Under Value or Over Value at each point
* Excel priority file:
  + Used to compare output system information to find the fix and failure data

3.2.1.3 Processing

* Processing is done by our main parsing class which gathers:
  + 1st pass percentage
  + Failed Test points
  + Config system information
  + Priority repair information

3.2.1.4 Outputs

* Graphical User Interface showing:
  + 1st pass percentage color coded
  + Scrolling log of failed test points and their fixes

3.2.1.5 Error Handling

* The only error handling we had was a check for the correct config file upon first run. This checks to make sure the number of test points on the config file match the number on the .rdy file.

## **3.3 Use Cases**

### **3.3.1 Use Case #1**

Replace Pogo Pins

While the software is running on a functional testing machine, a failure occurs while the machine is testing a part. The failure is labeled imot\_FP9 and is described as a motor current issue. The test engineer recognizes the failure and uses the software to reference a solution. First he will Check/Replace the Pogo Pins for ECU, then Check/Replace Pogo Pins for connecting station to nest and if neither of those work he will Contact the Process Tech/Engineer

### **3.3.2 Use Case #2**

Part Change

There has been a change of part assembly in the line. The test engineers recognize that the software is not designed to automatically react to this change and attempt to resync the software. The software recognizes that the parts coming in do not match the configuration and priority file as the previous part and prompts the engineer to choose a file. The test engineer loads the proper files and the software resumes the normal tests with the new configuration for the unit.

## **3.4 Classes / Objects**

### **3.4.1 TSD\_Main**

The TSD Main initializes the TSD Parser class and contains a while loop to check for new RDY files, getting new RDY files, updating the graphics class, and deleting the RDY file.

Attributes:

tsd: TSD\_Parser Object

Functions:

None

3.4.1.1 Attributes

3.4.1.2 Functions

<Reference to functional requirements and/or use cases>

### **3.4.2 TSD\_Parser**

The TSD\_Parser class does functions that are associated with RDY files, config file, and priority files. It extracts the names and data of each run recorded in the RDY file.

Attributes:

Tsd\_record: TSD\_Record Object

rdyFile: String file location of first RDY file

partNumber: String Part Number extracted from the first RDY file

Folder: String file location of RDY folder

excelFile: String file location of the config excel file

priorityFile: String file location of the priority excel file

failList: Array of strings containing failures that correspond to line numbers in RDY file

init: array of tkinter objects that initialize the tkinter window, menu buttons, etc.

Machine1: TSD\_Graphics object displaying data for line 62

Machine2: TSD\_Graphics object displaying data for line 64

Functions:

checkNewRDY: No Parameters. Checks the designated folder location for RDY files for a new RDY file. Returns true if files are present, false if folder is empty

getNewRDY: No Parameters. Sets the self.rdyFile variable to the location of the latest RDY file in the folder. Returns nothing.

delRDY: No Parameters. Deletes the checked RDY file from the folder. Returns nothing.

getNumLines: No Parameters. Checks the self.rdyFile for how many lines the text file contains. Returns the number of lines minus the header.

getExcelRows: No Parameters. Checks the self.excelFile for the number of test points it contains. Returns the number of test points.

getMachineName: No Parameters. Parses the machine name from self.rdyFile. Returns the machine name as a string.

getpassFail: No Parameters. Parses the self.rdyFile for the first pass pass/fail value. Returns a value from 1 to 4.

getPartNumber: No Parameters. Parses the self.rdyFile for for the part number being tested. Returns a string for the part number.

checkTestPoints: No Parameters. Parses the self.rdyFile for any test points that fail. Returns an array of line numbers that contain failing values.

replaceExcel: No Parameters. Opens up the file explorer and lets the user pick a new config file. Replaces the self.excelFile with the new file. Returns nothing.

update: No Parameters. Calls checkTestPoints and updates the graphics objects with the passFail values. Returns nothing.

### **3.4.3 TSD\_Record**

The TSD\_Record saves the percentages to a text file of each machine each time an update is called.

Attributes:

machine1Name: String name of the machine name.

partNumber1: String part number being tested.

percentage1: String percentage of the machine.

Time1: String timestamp when the update is called.

machine2Name: String name of the machine name.

partNumber2: String part number being tested.

percentage2: String percentage of the machine.

Time2: String timestamp when the update is called.

Functions:

init: Parameters are the attributes listed above excluding time. Sets the arguments to instance variables. Writes the values into the text file. Returns nothing.

updateRecord1: Parameters are machine name, part number, and percentage. Sets the instance variables to the passed arguments and writes to the text file. Returns nothing.

updateRecord2: Parameters are machine name, part number, and percentage. Sets the instance variables to the passed arguments and writes to the text file. Returns nothing.

### **3.4.4 TSD\_Email**

The purpose of the Email class is to set variables that will allow win32com to assign the specific information needed to send the email.

Attributes: None

Functions:

send\_email: Parameters are the machine name and value. The method sends an email to the designated “TO” field. Returns nothing.

### **3.4.5 TSD\_Graphics**

The purpose of the Graphics class is to take the information extracted by the Parser class, calculate the percentage, and display it graphically on the screen. The Graphics class is responsible for showing the first pass percentage as well as a corresponding color depending on threshold values. It will display the machine names, current part, number of runs, last run values, list of individual test point failures and fixes, and a line graph showing the progression of the percentage.

Attributes:

root: tkinter object to initialize window.

Menubar: tkinter object for managing window menubar.

cCanvas: tkinter widget to allow drawing. The topmost portion holding the circles.

lCanvas: tkinter widget to allow drawing. The bottommost portion.

failFrame: tkinter frame to hold the treeview showing the failing parts and recommended fixes.

lineName: String name to hold the name of the machine.

currPart: String name to hold the name of the current part being tested.

x: vertical offset of each machine display on the canvas.

y: horizontal offset of each machine display on the canvas.

Val: Float value holding the current percentage of the runs.

Count: The number of rdy files parsed.

totalPass: Number of tests that have passed.

totalFail: Number of tests that have failed.

startTime: Double value holding the current time when the graphics class was initialized.

valArr: Array holding the last 200 percentage values.

lineArr: Array holding tkinter line objects, representing the line graph.

circle1: tkinter circle object graphically showing first pass percentage using color.

circle2: tkinter circle object graphically showing last run percentage using color.

firstPassLabel: tkinter text object displaying the last run details.

failList: Array of the failures that have been encountered.

percentDraw: tkinter text object that will display the percentage.

lineNameDraw: tkinter text object displaying the machine name.

lastPartName: tkinter text object displaying the last run details.

lastPartPercent: tkinter text object displaying the percentage of the last run.

Functions:

init: Takes in a graphics init array, a String machine name, a String current part name, a String current Part Name, and integer x, y values. The graphics init array is an array consisting of the tkinter window, menubar, two canvases to draw the circles and line graph, and a frame holding the table of failing parts. All of these objects are shared by two instances of the graphics class.The labels and shapes are initialized and placed in their appropriate x y coordinates. Returns nothing.

update: takes in an integer passfail value, String representing the current part number, and a string of the highest priority failure of the current rdy file. The display does not show the percentage nor the circle will change color until 5 runs have passed through each machine. If the threshold is reached at some point, the email class is called. The passfail value is used to increment totalPass or totalFail, depending on the number. Count is incremented each time. The percentage is calculated. The first value in the valArry array is removed, and the current percentage is appended on. The line graph is redrawn. The circles and labels are redrawn. Checks if a failure is passed in the parameters, if it is, it is appended to the treeview. Returns nothing.

getName: Takes no parameters. Returns the name of the machine.

getPercentage: Takes no parameters. Returns the current percentage of the line.

## **3.5 Non-Functional Requirements**

### **3.5.1 Performance**

The program should be able to process one rdy file within 15 seconds. Text and colors from the GUI must be legible from a distance of 30 feet from a 20-inch monitor. The program should be an accurate measure of how test stand is performing.

### **3.5.2 Reliability**

The program should not crash due to any memory leaks. The system should be able to run for days without a restart and handle thousands of runs.

**3.5.3 Availability**

The program should not be dependent on any online services. It should be a standalone product that will require no downtime.

### **3.5.4 Security**

Data should not be shared online from this product. It should only write percentages, machines names, and passfail values in text files locally stored in the machine running the program.

### **3.5.5 Maintainability**

The program should be well-documented to allow future developers from Bosch to edit and add new features to the program. If any changes to the specification occurs, such as new rdy file formats, etc; a test engineer must be able to determine which values to change to allow the system adapt to the new specifications.

### **3.5.6 Portability**

The program should have an installer that will allow it to install on any windows machine with internet connection. It will install any dependencies the program needs. The system can be used on a laptop connected wirelessly to Bosch’s network, accessing its network drives.

## **3.6 Inverse Requirements**

This program is not intended to allow anyone to able to fix the test stands by themselves. It only shows the recommended fix for each part failure.

The system may not be running real time. The timing will depend on how quickly RDY files are processed and copied over to the designated folder.

## **3.7 Design Constraints**

The GUI should be a simple interface that is highly legible from a distance. It does not have to be visually pleasing; it will need to be practical.

The program should be written in an easy programming language that test engineers can understand, and allow for future improvements and modifications.

## **3.8 Logical Database Requirements**

A database is involved with this system, however, it only relies on copying at outputted file from the database before it is deleted. An RDY file is outputted by Promaster and a script was written to copy the RDY file to the designated network folder. No direct access to the database is involved in this process.

## **3.9 Other Requirements**

The config excel file listing all the part names and the priority files listing the priority values on the part failures should have matching part names. The priority excel file part names may have a TS\_- prefix.

The rdy file and the config file must have the same number of test points.

The text files holding test data must be in .RDY format.

The config and priority excel files must be in the .xls format.

The first pass-fail value should be line 16 in the rdy file.

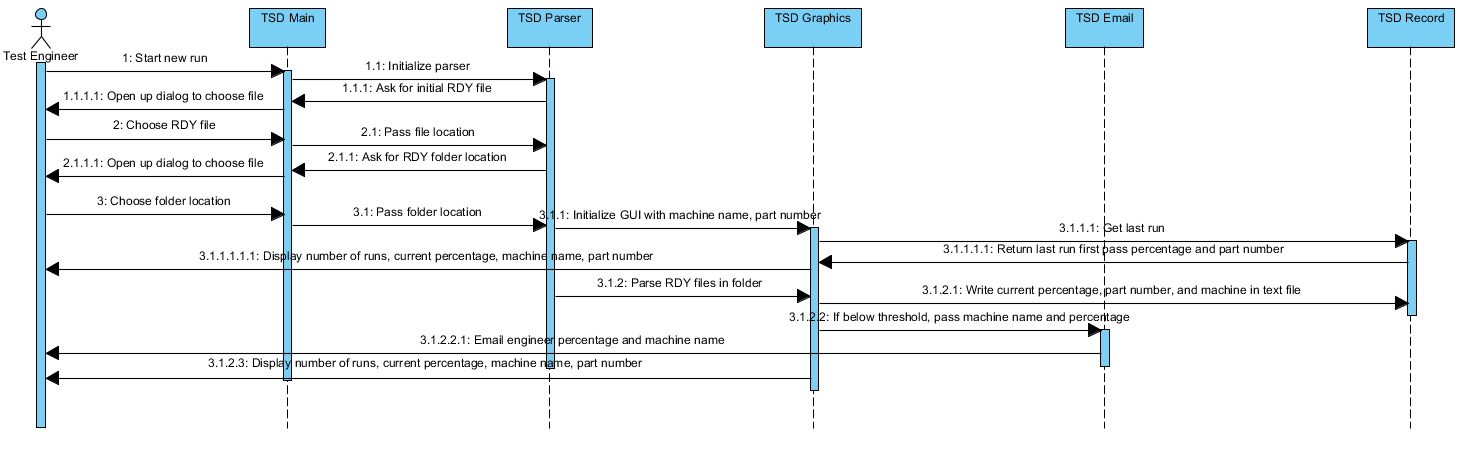
The machine name should be line 8 in the rdy file.

The current part number being tested should be line 5 in the rdy file.

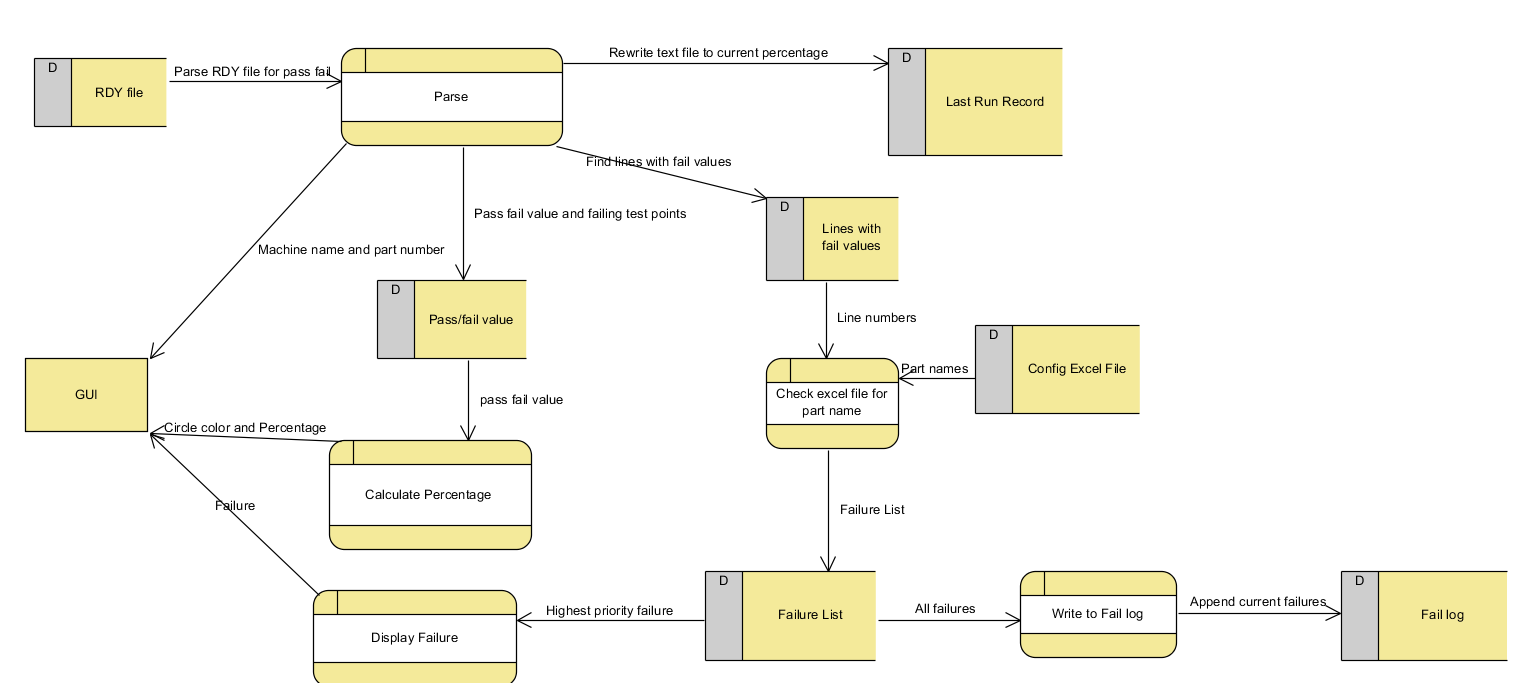
The test points should start at line 43, with each test point having 6 values in between in the rdy file.

# **4. Analysis Models**

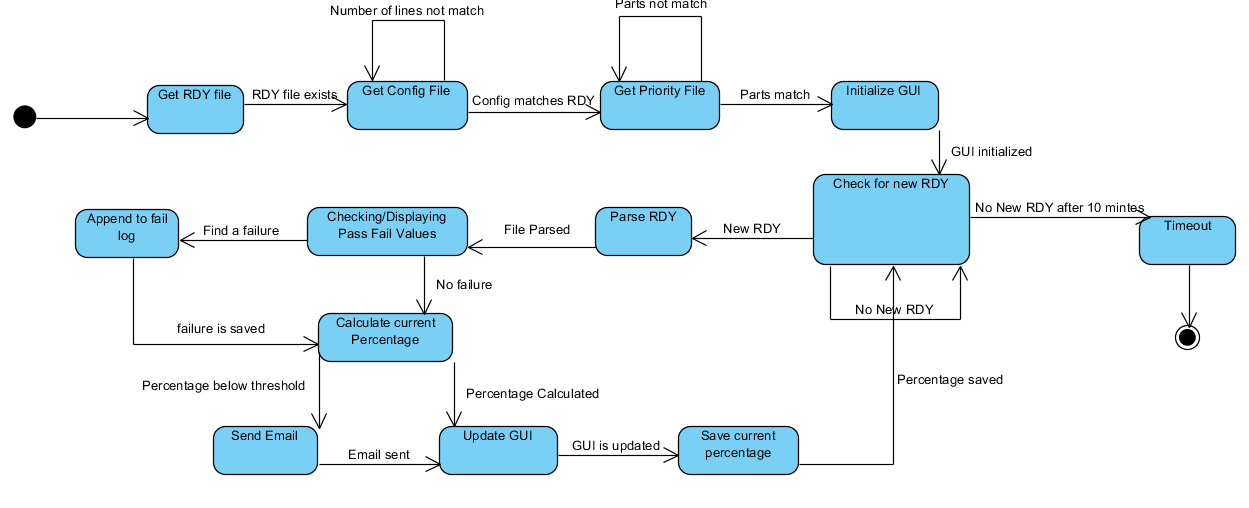
## **4.1 Sequence Diagrams**

g

## **4.3 Data Flow Diagrams (DFD)**



## **4.2 State-Transition Diagrams (STD)**



# **5. Change Management Process**

Once this document has been completed by this team of developers, the entire project will be given to Bosch. The test engineers will have free reign to use this software as they please. This may be the only management change that will occur in its lifetime.

# **A. Appendices**

## **A.1 Appendix 1**

None

## **A.2 Appendix 2**

None