Nathan Hightower

Project Documentation

A check mark on a table

Description automatically generated

# Executive Summary

The proposed project addresses a need in the production sector within our company. Currently, we have a variety of autoclave and non-autoclave cures. These all are set to their own specific requirements according to the ASC-PRS documents. As it stands Quality Engineering/Material Processes Engineering has to review these cure reports mostly manually utilizing some basic excel formulas to find ramp rates, temperature thresholds, deltas, vacuum pressure, hold times as well as compliance to the 9002 document. Depending on the cure being utilized it can generate upwards of a few hundred thousand data points to review at the expense of a couple of days and off shift support. This is a rather excruciating process for these groups and all of them have expressed their desire for something to help alleviate this burden.

This is where what I call Curefy (Cure-Verify) comes into play. This will be a piece of software designed specifically for the task above. The overall idea is as follows:

1. Opens a user interface (UI)
2. This UI will have one drop down box to select the ACS-PRS spec to review the data too
3. The UI will have a box to search for and import the .xlsx file where the cure data is stored
4. The UI will have the normal close x in the upper right-hand corner
5. The UI will have a generate report button to run the verification process

Curefy will generate a report based on the imported data, the ACS-PRS documents as well as the 9002 document. This report will pull all TC’s (Thermocouples) that fail at each step of the cure process/ including the time at which each failed, TC name and the data recorded at time of failure. This could look like this:

Step A – First Hold – Exceeded Max Temperature

TC1: 00:24:00 – 02:24:00 – Temp Max: 176 Min: 166

TC2: 00:24:00 – 02:24:00 – Temp Max: 186 Min: 155

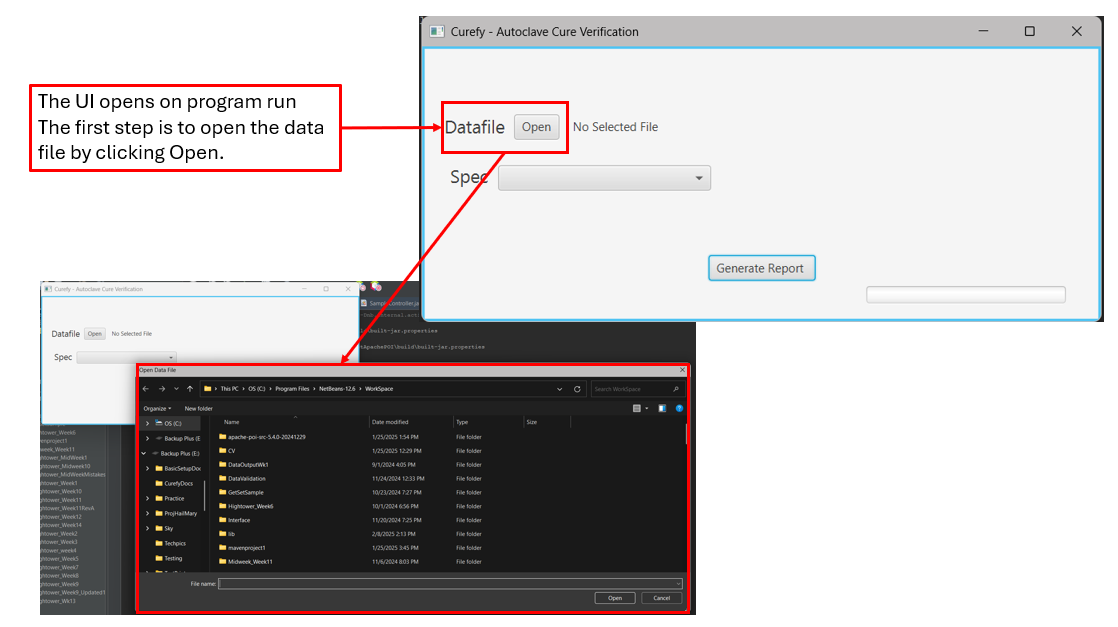
This could be the format through the remaining steps.

The specifications (ACS-PRS) / 9002 can be stored simply by excel. When selecting within the UI we will be telling the software to use this specification by looking in the stored workbook for the specific sheet and data stored on that sheet to compare the imported data too.

The report will be displayed after running the software allowing the user to save it for later use.

Initially, this will be a limited use application. We will scale down to incorporate one or two specifications to allow us to create a working example. This example will be reviewed for effectiveness and accuracy at which point we can start including the rest of the specifications.

# Scenario | Program Usage Description



# Scenario | Program Usage Description

A screenshot of a computer

AI-generated content may be incorrect.

# Scenario | Program Usage Description

A screenshot of a computer

AI-generated content may be incorrect.

# Scenario | Program Usage Description

A screenshot of a computer

AI-generated content may be incorrect.

# Scenario | Program Usage Description

A screenshot of a computer

AI-generated content may be incorrect.

# Reports

The report utilizes excel to store the information found by the program as seen above. It has the following tabs: Header, TempOOTs, RampRatesOOTs, VacuumOOTs, 9002OOTs. Each tab holds its own specific data. Header and TempOOts seen here.

# A screenshot of a computer AI-generated content may be incorrect. A screenshot of a computer AI-generated content may be incorrect.

# System Architecture

A screenshot of a computer

AI-generated content may be incorrect.

## Source Code Structure

Source code structure introduction. The following is a summary of the source code directories and their contents:

|  |  |
| --- | --- |
| **Code Directory** | |
| **Directory** | **Usage** |
| Curefy.java | Sets the scene for the UI and launches it for use |
| SampleContoller.java | Uses the UI to select datafile, cure specification and generate the report by running the cure |
| sample.fxml | Style sheet for the UI |
| CureSpec.java | Used to grab the curespec data and assign that to variables to be used |
| ExcelHandler.java | Houses all methods to review the data per spec, write data to report, and open report file once complete |
|  | |

# Executables

Describe the executables that are a part of this project. If there are multiple executables break out each one into its own heading and give its name and a description of what function it plays in the system.

**N/A**

# Code Architecture

The data store is an excel file or actually multiple excel files. The code accesses them using Apache POI libraries to locate specific sheets or points in those sheets for data to be used or for placing reviewed data into.

## Database or Data Store

The program ask for the cure spec data (right), then ask for the cure data(bottom right), and after review it generates the report (bottom left).A screenshot of a spreadsheet

AI-generated content may be incorrect.

External Files & Data

The external files include all .xlsx files. Cure spec, report file and cure data file itself as seen below and included in the GitHub repository

A screenshot of a spreadsheet

AI-generated content may be incorrect.

Programming Language | C#.NET

The program is entirely written in Java. This is not to include the .fxml file which uses something similar to that of a CSS style sheet but used java to generate that.

Project Classes

Classes within the project are used to abstract re-usable pieces of code. Classes are also used to group related values, known as properties. The project utilizes these classes:

### Short Class Description | Curefy.java

Main method to set and launch the UI

### Short Class2 Description | SampleController.java

To handle all UI interaction and begin cure review process

### Short Class3 Description | CureSpec.java

Creates a cure spec from the data file to use during program run

### Short Class4 Description | ExcelHandler.java

To preform all review and reporting methods

Summary

The program uses the Java language. It relies on the Apache POI and JavaFx Libraries which need to be included in the classpath to work. I wrote it using Netbeans IDE. There are three files required to run this program that is the cure spec, the data file, and the report template. All of which are .xslx files and are included within the GitHub repository. The program will run the UI that will ask for these files and then we can run the program and view the report once completed. There are four classes as of right now and one fxml file used as a style sheet for the UI. The data store option in this example are the excel files that have the data needed for the reviewing process.

# APPENDIX B (BUILD AND RELEASE PROCESS)

The update process will use two instances of the program one for testing and the other active. When updates need to be made we will test in the testing version before releasing into the active system.

# APPENDIX C (CLIENT INSTALLATION INSTRUCTIONS)

This project will need to be packaged before a client machine is ready to use this. That is something to come and is not a priority at this given moment. There is a year worth of work to be done before this comes into play.

# APPENDIX D (DEVELOPER SETUP INSTRUCTIONS)

The developer would just need access to an appropriate IDE with the correct libraries added as documented within this report.