LAFAYETTE COLLEGE

Maintainability Program Plan for:

LFEV – SP17 – Y5

Version: 0.3

Submission Date: 3/1/2017

File Name:	Maintainability Program Plan Template	
File Location:	VSCADA Git repo	
Version Number:	0.3	
Created By:	Martin Townley	02/10/2017
Reviewed By:		
Modified By:	Kemal Dilsiz	02/28/17
	Marty Townley	02/28/17
	Austin Wiles	02/28/17
	Craig Lombardo	02/28/17
	Raji Birru	02/21/17
Approved By:		

Table of Contents

1. Overview

1.1 Scope

This maintainability plan describes the necessary tasks, responsibilities, and controls that should be implemented in the Lafayette Formula Electric Vehicle Project.

The primary function of the maintainability plan effort is to document the procedures; ensure both high operational readiness and availability; and ensure the ease in transition of the project from year to year.

1.2 Purpose

The purpose of the maintainability plan is to:

- Define the maintainability tasks to be accomplished
- Ensure software design decisions are made with maintainability in mind
- Assess maintainability trade-offs that will be encountered.
- Open inter-team discussions with regards to maintainability approaches

2. General Requirements

The vehicle for commitment to effective R&M engineering is the R&M program plan developed for the project. The RMPP should emphasize early participation commencing with requirements definition and system development, followed by a comprehensive test, corrective action, and demonstration program to identify and correct deficiencies as required. The RMPP should be implemented at the onset of a development and subcontractor/vendor selection process.

3. VSCADA

3.1 Software Installation on Hardware

SCADA software shall be run via a Java Virtual Machine (JVM). Any Linux system supporting the Java Development Kit 8 (JDK) shall have the capabilities to run our software. We are confident in the functionality as a Java class file is a file containing Java bytecode which is

executed on a JVM. Proper setup of the latest version of Java shall allow for execution of such bytecode. Installation scripts shall run to verify software dependencies and create a working database. All dependencies, such as external Java libraries, packages, etc. shall be provided in the download file. There will additionally be an included README file that will outline how the software is supposed to operate. Updates will be done manually so the user may verify all systems check out. Automatic updates are not in the scope of this project. To manually update, use SCP or a USB drive to move the files to the hardware and then manually compile the new or changed files using the makefile provided.

3.2 Errors, Exceptions and Logs

Errors and exceptions are handled via an error table. Errors will be categorized based on a criticality level, depending on the criticality, an error can be “non-critical”, “advisory” or “critical”. “Non-critical” errors will only be logged into the database and will not generate any warnings for the user. “Advisory” errors will trigger a popup warning on the display after a certain level of criticality has been reached. “Critical” errors will directly cause a message and potentially halt running systems should the safety of a user be put in danger. If the safety of the user is at risk, the SCADA system shall trip the safety loop and notify the user of the system that caused the fault.

When there are either no, or only “non-critical”, errors the system will be in a “stable” state. The stable state will allow for full functionality of the system with no visible interruptions. If there are “advisory” warnings then the system will be in a “cautionary” state. Full functionality of the system will still be available; however, there will be warning messages present on all screens. Finally, if there are “critical” warnings then the system will move to an “unstable” state. The system itself shall not crash; however, there will be warnings present and the safety loop switch shall not be opened until all errors are resolved and the user presses the GLV driver resettable switch. There will be user defined parameters for what sensors are critical and over what value ranges those criticality levels are defined. These parameters will be configurable through a custom user interface in the application. It is the option of the user at this point to either resolve such errors or to move to drive demo mode in which reasonable values will be simulated for the missing systems.

Regardless of the system state, assuming the SCADA system is powered on, all error messages and subsystem data will be accessible via the SCADA server, as viewed on either a computer, tablet or phone. The user will not need to manually create SQL queries, but rather with the application launched, select the information they are looking for. Users will be able to

select the desired subsystem, or subsystems, as well as the date, or date range, that they are looking for. Data will be viewable through the application, and will be available for download. Downloads will include either exports in the form or CSV or database files. CSV files will be useful for simple manipulation and a database file will be useful for manual SQL querying or imports into MATLAB using the “fetch” command.

3.3 Backup and Restoring

Backups will be performed by default after 50,000 data points have been logged to the system. This includes sensor data, error messages, etc. Upon completion of 10 backup cycles, the system will port all data to a CSV file and compressed to a ZIP file for long term storage. Porting to a CSV will allow for minor compression and then compressing it to a ZIP file will allow for large scale compression. Users shall have the ability to change any and all of these configuration parameters, whether it be the number of data points before backup, number of cycles, auto-trimming, when to erase old data, etc. Restoring databases shall be done automatically. In the event of a sudden shutdown, the system shall be able to restore to the last stable version of the database. The data being inserted at that point will very likely be lost; however, backups shall help prevent any chance of data loss.

Users will have the ability to insert a media device to export all of their configuration files for use in installation of SCADA systems on new devices. If a user intentionally, or unintentionally, nukes the entire system and there are no backup files saved on an external device then the user must go through the install procedures on new hardware. This is due to the fact that we are not supporting methods for remotely/automatically saving configuration files via software such as GIT or SVN.

3.4 Deployment on New Hardware

The deployment process for new hardware is simple. The SCADA software runs on the JVM. As long as the operating system is a Linux distribution that supports Java, the software will run. Full documentation for manual installation of software will be provided, with installation of all dependencies used. This will include versions of all dependencies. In addition to this, we will provide a script that runs the installation and verification process on known hardware configurations (currently Raspberry Pi 3). This script will be only useful for our version of SCADA software. If future teams decide to change it, then the script is not guaranteed to work as intended. There will be an included README file that will outline how the software is supposed

to operate. At startup, the software will run automatically. When the SCADA system is closed or crashes, the exit code will be noted and depending on the exit code, the software will have the capability to be automatically restarted. The exit code for a crash is different than the exit code of a clean exit (say a user exits the software on purpose). If specified by a configuration file, when a crash is detected, we can restart the software.

3.5 Log File Trimming

Log files shall be removed from the database similarly to the methodology used for backups. A user specified parameter shall control the number of logs that are retained before the database automatically trims itself. The user shall have the opportunity to turn this feature off; however, that will require the user themselves to go into the configuration interface and manually trim data as necessary. The configuration interface will allow the user to select a date range and the system(s) that they would like to trim the data for. They will be able to delete specific ranges of data or all of the data logs. While space is not a major concern for us, it is something to note as the cleanliness of the database may diminish.

3.6 System API

We will provide a web API to interact with the system and the database (by one week after CDR). The web api will allow future users to extract information about the car, such as status of individual subsystems, and will also allow users to configure new subsystems. Having functions that allow future users to easily write extensions or other applications while not having to touch the VSCADA code is our goal. An example of an extension that will be interacting with our API is the Cell Phone App. The Cell App team will be calling paths on the web server, and the web server will then call on functions in the VSCADA code to perform the expected task. Through our API, we plan on allowing users to extract all previous data (that is stored), retrieve only the most recent packet of data, retrieve data between certain times, retrieve data for specific subsystems, and to either add, delete or edit subsystems. Providing users with as much functionality possible without any need to touch the code itself will help support future project teams. The more functionality we provide, the less they must create.

3.7 System Configuration Maintainability

The system will detect changes in hardware configurations on startup using CANbus communications. The only extra configuration needed is if an entirely new subsystem is being installed (for example, an extra pack). This will be taken care of in the API, and no new code must be written to extend the system in such a way. If the new subsystem is configured incorrectly, it's data will not be stored. In order for the main tables in the database to function properly the IDs sent over CAN must map to a sensor in the main SensorLabel table (an example is shown in Figure 1.) Data will be stored in a simple table (an example is shown in Figure 2.); however, without an ID in the SensorLabel table to join on, that data has no meaning so is effectively useless. The database is the center of everything. Upon configuring a subsystem, with a unique ID, that unique ID will be the CANBus ID used for communications so we can keep track of subsystems. When we receive data over CANBus, we put the received data along with the ID and an auto generated TimeStamp into the database. This will make extraction of data simple. The ID of present subsystems will be changeable via a configuration interface.

The software will continue to function when only some of the system hardware is available. There will be notifications about which subsystems are missing, and the status of available subsystems. If a subsystem is missing, we will be able to simulate the data from that subsystem so the rest of the car can be tested in various configurations. A "Drive Demo mode" will be provided to give the users control over the data being simulated. They will be able to change the simulated data on the go without need to recompile.

ID	sensorName	sensorUnits	dataType	system
1	cellV1	V	FLOAT	TSV
2	cellV2	V	FLOAT	TSV
.	.	.	.	.
.	.	.	.	.

Figure 1.
SensorLabels Example Table

sensorID	value	TimeStamp
1	2.30	2017-02-27 16:31:40
2	2.28	2017-02-27 16:31:40
.	.	.
.	.	.

Figure 2.
Data Example Table

3.8 System Configuration Checking

The system configuration is checked once upon startup. The software will run a systems check and alert the user if any systems are missing. A subsystem can be attached dynamically without restarting, as we will be examining all information coming in on the CANBus. When a missing subsystem is attached and we begin receiving packets over the CAN, the software will act as though the system has been there all along. If the check returns that too many vital systems are missing, the car will transition to different system states as specified in section 3.2. Regardless of the system state, and assuming the system is powered on, we will be able to view the system in maintenance mode. If any of the systems are "silent" for 30 or more seconds, an error will be logged and the user will be notified. To achieve this we will have a watchdog which will help us in detecting "silent" systems. "Silent" systems, and their importance, will be determined by the same criticality table as mentioned in section 3.2. Custom configuration files will be scanned for initially; however, there will be a master configuration that will be hardcoded into the system that will only be read if it cannot find a valid user configuration file.

3.9 Tool Chain, Design Suite

The SCADA system is compiled using the JDK 8 compiler, as it is the most up to date and stable Java release. We currently have no implementation for the debugger; however, installation of the JDK will allow proper integration for such use. All SCADA software is to be run in a JRE, in the current case JRE 8. All source files, libraries, and any other dependencies that will be required for use with the SCADA system will be included in the program package and will be installed as the program is being installed on a new environment. While it is not the goal of our team to require superfluous downloads, as mentioned in the last point, if there are dependencies that are not found but are required for core functionality of the SCADA system, they will be automatically installed during the installation process.

3.10 Third Party Software

The code for the SCADA system is being written in Java (Java 8). All code written must compile in Java 8. In previous years, Python was used. This did not take advantage of multiple

cores and would max out a single cpu core. The switch to Java is to have more control over threading and use of cores. The front-end user interface views will be written in Java 8 using JavaFX. The web server framework we are using to communicate with external machines is Spark (http://sparkjava.com/). Spark was chosen for use by our team as it provides users with an easy to use micro-framework which was designed for Java 8. We will be using JSON as our universal format of choice. We are levaraging Google's GSON library (https://github.com/google/gson) to convert Java Objects to JSON for easy and rapid communication.

We will be using SQLite as our database to store data. We have verified that SQLite offers the capability to backup data for protection, as well as restore a backup should there be a sudden failure such as a power disconnect. If there is a sudden and unexpected shutdown the database will not become corrupt. We can back this claim with information from the SQLite site (http://www.sqlite.org/transactional.html). "All changes within a single transaction in SQLite either occur completely or not at all, even if the act of writing the change out to the disk is interrupted by: a program crash, an operating system crash, or a power failure." This has been "extensively checked in the SQLite regression test suite using a special test harness that simulates the effects on a database file of operating system crashes and power failures." It is for these reasons that we are certain that an SQLite database will be suitable for the purposes of SCADA.

3.11 Requirements of GPR007

a. All code shall be maintained on the group GitHub with snapshots and final released code available on the website. All code on the Git shall be commented and the use of version release shall be used to easily distinguish between major and nominal software changes. Release snapshots will be released on the website and the DVD.
b. There will be no login screen to the SCADA system. A fully installed system shall have the capability to start up in a desired mode without the interaction of a user. SCADA shall not have any ability to turn on via timers, the cell phone app, or any other remote protocol, a user must physically turn the system on. The only login that will be required will be for use of the configuration user interface. We want to ensure that a random user cannot go in and alter key tables used by the SCADA system. The password will be well documented and easily available.
c. The installation of SCADA software on a new PI, or computer, shall be made simple for Linux users. There will be a downloadable file with a Makefile included in it as well as a

Readme. The Readme shall provide documentation for installing and troubleshooting; however, the user should only have to run commands such as “make install” or “make clean” etc.

d. There shall be a default set of system parameters for any aspect of the SCADA system that is reconfigurable. Users will be able to launch our ConfigurationUI which will allow them to create, edit or change the configuration table that will be referenced by much of the SCADA software. Through editing this table, there will be no need for recompiling as it is merely a reference point.
e. All data will be logged in SQLite databases (https://www.sqlite.org/). It is a very easy to use but powerful tool with many different strengths. SCADA, as well as the cell team will have a universal format of JSON. We have decided to make this our universal format as it is well recognized, easy to read, easy to learn, scalable, and versatile.
f. USB support will be available for file transfer. USB devices shall be detected automatically. Users will be able to transfer database information such as backup files, program files etc. Section 3.2 addresses the means for which data may be ported out. A user may also copy all of the content in the file system.

4. Cell Team

4.1 Software Installation on Hardware

The Android application will be packaged in an Android application package (APK) and this format allows the user to easily install the application to an Android based operating system. This apk file only needs to be transferred to the tablet/phone and the android system will install it if the file is started.

4.2 Errors, exceptions and logs

Errors can be categorized into two groups for the cell application: the errors in the application and the errors in the system that come through the available data from SCADA.

Errors in the cell phone application are handled through the Android system if it is anything that affects the display or if there is an error in the code. In many situations, a simple restart to the application would solve the problem. In addition to this, the application will create “application specific” logs for these kinds of errors which can be used to fix the code by the future teams.

Errors caused by the system will not necessarily affect the cell phone application since it is a separate module. The data that comes to the application will be “read only” and there will not be any way to interfere with anything related to an error in the system. If an error occurs in the system, and if SCADA team cannot handle the error situation well and the cell phone application can receive damaged or wrong data. Damaged(incomplete) data will simply create a notification indicating the error situation whereas the wrong data would simply be passed to the displays. Therefore, if a wrong data is passed to the cell phone application then it would be difficult to detect it. However, no data is being stored by the cell phone application and this would not change anything for the application.

Another error situation could happen while mobile application requests data from the SCADA team for a package of data. If the cell phone application for some reason sends requests without stopping, and spams the server that SCADA has, then this could possibly create a problem for the SCADA team. This specific situation will be tested by both ends throughly to ensure that the mobile application doesn’t affect the SCADA team. And there will be a safety shutdown feature in case the application gets out of control in terms of sending requests.

Logs are another important aspect of the application. There will be two kinds of logs, the logs that are stored in the device for the mobile application and the logs that are sent through the SCADA system for the updates on the whole system. The logs from the application will be stored locally and the logs from the SCADA team will strictly come through the web server connection. Both will be available as a view in the display options and the user can choose to see them whenever they would like.

Exceptions in the general system will be handled with threshold checks and notifications. Again similar to the errors, the exceptions will not directly affect the application since it is a separate module to the whole system. Exceptions in the system will alert the user and cause a vibration and will also check for certain inputs in the log messages provided by the SCADA team. If the application finds an exception when there isn’t one, the user can simply ignore the notification because the application will keep running without any changes.

Exceptions in the software will be tested thoroughly with unit testing but in case of an exception or need of a modification, the code will be on a version control platform and can be fixed by a future team.

4.3 Backup and Restoring

The software will generate the views every time it is launched. This makes sure that the application will not need a continuously running function which requires constant backup. However, the versions of the software will be stored in a version control platform. This makes it possible for future teams to create an apk for a different version of the software and be able to experiment with branching. Therefore, the backup will be done while coding through version control and the restore procedure will be deploying an apk of a different version.

4.4 Deployment on new Hardware

Through the apk file.

4.5 Log file trimming

Log files will be automatically trimmed and will store upto 5mb of data. This is a ridiculous amount of space for the simple logging that the application will do. If wanted, the log files can manually be copied from the android device to another storage. An automatic process for saving the log files beyond 5 mb currently is a low priority of the design.

Each character takes exactly 8 bits to be stored in, that's 1 byte (B). so 1 MB = 1024KB and 1 KB = 1024B so 1 MB has 1024 x 1024 = 1,048,576 bytes, that's 1,048,576 characters, that includes letters, spaces, symbols, etc. This is approximately 500 pages of text for 1 mb. If we spare 5mb for the logging, that is roughly 2500 pages of logs.

We will separate the log text files into smaller parts so that the system does not have to load all the log messages for generating a display.

4.6 System API

The system will have a graphical user interface generated through the android studio tools. We hope to build an innovative environment for users where they can generate a look special to their need. Configuration of the API is one of the biggest aims of the application.

First of all, we want to make it possible for the application to run both on tablets and phones smoothly without any problems. There are multiple tools in Android Studio to make this

possible. The android layout tools makes it possible to have multiple displays for different situations. For differently sized screens, different displays can be generated.

Also, it is possible to use “dp” for size of the layout. Dp makes it possible to calculate a certain size according to the relative size of the screen. Therefore, a chart can take 50% of the screen size with only one layout code.

These two tools makes it possible to generate displays that will support not only both tablets and mobile devices but at the same time it will make it possible to generate displays that are similar in all devices. Additionally, we hope to make it possible to zoom in the layout.

Another very important aspect of the API is the expandability. If we were to add more components to the LFEV system, how will the cell application update the display accordingly. Will it be able to adapt?

For this situation, we decided to make it possible for the user to generate their own displays. The incoming data will be stored in a certain way to make it possible to generate your own displays according to the options made available by us.

The user can choose their x-axis and y-axis together with adjustable time for a specific chart. The use can generate a speedometer and decide on the lower and higher limit and see simultaneous update. The user can choose to see the log messages according to a certain time.

Finally, another feature we are hoping to implement will make this application API much more flexible. This is to be able to generate your own display. Instead of user displaying a single part of the data, for example only one chart of Speed vs Time, the user can create their own displays. For instance, on a tablet, the user can first make a chart of SOC vs time and put this chart on top left of the display. Then the user could create a speedometer and put it on bottom left. And lastly, they could display the log messages on the right of the screen. Such generation of displays would give the user full control over what they want to see. And if new components are added to the system, new system API possibilities would be generated automatically by the application.

4.7 System Configuration Maintainability

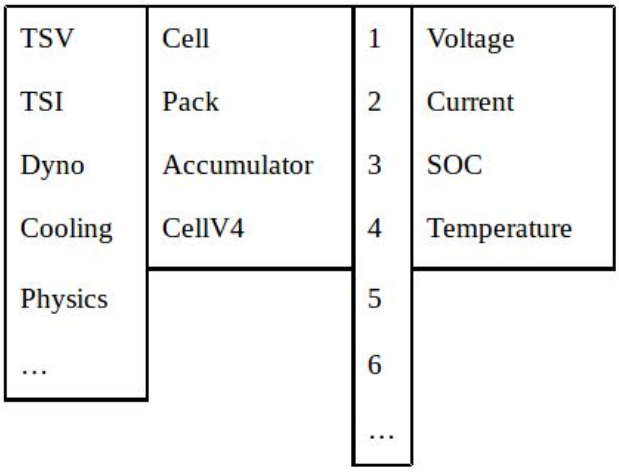
The system configuration will be in two major steps. First step will be how the system will adapt to the changes in the hardware and the transfer of the data. This will make it possible for the application to generate new displays without ever needing to change the code. The second part of the maintainability plan is concerned with the software itself. If one wishes to improve the software or make new display options and such, how will the application design allow this.

Firstly, I will provide a rough example of how the data will be packaged and how will this make it possible to generate new displays without needing to change the application.

ID	sName	sUnit	sSystem
1	Cell	V	TSV
2	Cell	V	TSV
3	Cell	V	TSV
4	CellV4	V	TSV
5	Pack	V	TSV
6	Torque	Nm	Dyno

```
{"subName":"TSV",
      "subSensors":{
      "1":{"sName":"Cell","sVal":"1.1","sUnits":"V"},
      "2":{"sName":"Cell","sVal":"3.7","sUnits":"V"},
      "3":{"sName":"Cell","sVal":"8.1","sUnits":"V"},
      "4":{"sName":"CellV4","sVal":"4.1","sUnits":"V"}
      }
}
```

The Json package on the top gives a good idea of how part of the package will contain the information. However, before we start discussing about this, I want to point out that there is a “sName” : “CellV4” and this is not a desired input by the cell phone application. This is added there to explain how the input into the SCADA database will affect the mobile application directly.



Here, System API will create a expanding tabs structure for the data display. This will be structured similar to the picture on the left. This shows the expanding tabs for “TSV → Cell → Any ID → Voltage”. As you can see the CellV4 is created separately because this was entered differently in the database by the SCADA user.

First of all, the application will group the inputs together according to the name provided. This will make the display for accessing data

simple. While generating these displays, the application will also create the data structures to hold the data for the current instance of the application. Also, the application will provide the user to create a dictionary to group together similar names. A simple example would be to say that any input of "V" will correspond to "Voltage" in the user interface. Thus, if the user decides to create a dictionary input of "CellV4" corresponding to "Cell", it will also group that particular input together with the other "Cell" inputs. This makes it possible for the user to create their own expanding tab displays together with the SCADA inputs.

If a future teams decides to add a "pack-5", this will automatically be available in the application through the json data provided. Such new data can be used to generate custom charts or other forms of display.

If the user, let's say, wants to generate a "TSV → Cell → 1 → Voltage" vs Time graph, the application will first generate the chart with the most recent information obtained from the SCADA web server. This information will be available in the data structures created together with the expanding display when the first batch of data is received. Later, the user can modify the time for the display of the chart/graph and this will request data from the SCADA server and instantly update the graph according to the requested data by the user.

When displaying a certain time period, the graph will not update itself but when the most recent information is on the display, there will be a certain sampling rate. This will be specified by the user, being at most every second, and the graph will update accordingly. If the user chooses every 5 seconds, the graph will display the data with 5 second intervals.

Another important aspect of this is that if a future team wants to implement a different type of database system, they will have to configure it so that it will pass through the application as a certain kind of object. This will make it easier for the maintainability of the system.

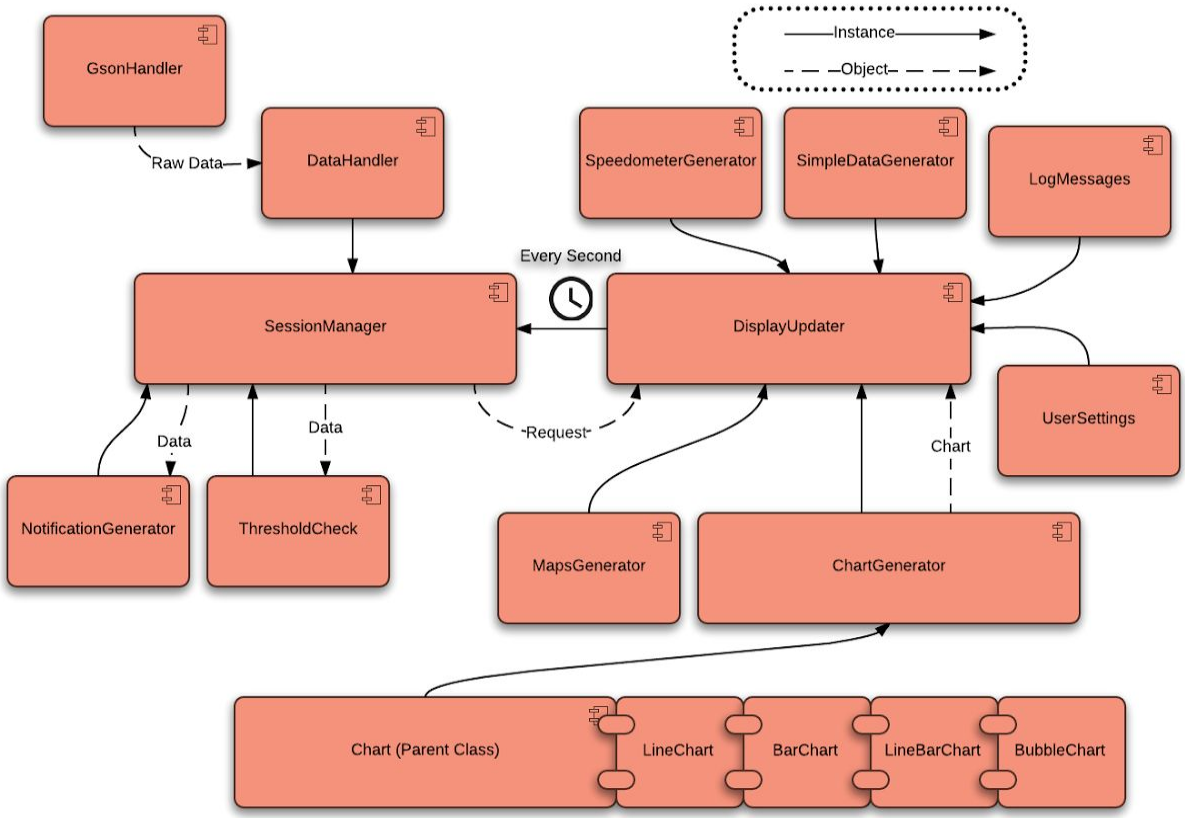
All this information will be documented with Javadoc as well as a tutorial prepared for the users and also a guide for the future teams about maintaining the application.

Secondly, another exciting feature of the application will be the easy possibility of expansion in the code of the software. Expansion in the software will be available thanks to the system design. Below is the content view from the design proposal. As you can see from the diagram, different View generators are sent to the SessionManager through the DisplayUpdater. This is a key part of the design as it will allow the future teams to easily add new Views for the system. The new Views can be easily added and passed through the Display Updater for the system.

Another expandability will be possible through the SessionManager. Currently, both software teams are planning to use Gson for data transfer to the android application. If in future,

another database is wished to be implemented, DataHandler will make it possible to pass any sort of data to the SessionManager. It is a medium between the preferred data transfer and the system.

Additionally, the NotificationGenerator and ThresholdCheck provides easy ways to add new notifications or threshold checks for the system. The application will send notifications in situations such as, SOC falls below 20% for a specific battery. The application will also send e-mails or messages or other forms of sharing features that will alert the user.



System Configuration Maintainability

The configuration will be specified in documents about the project and will be stored in version control. The system will run on any Android above version 4.0.3 and if future teams wish to increase the lowest requirement for the system, no changes should be necessary for the existing code.

The software requires a functioning android system to be able to run. There is no possibility for some of the functionality when the hardware is not complete. For the hardware, we are testing the parts of the software both in emulators and also in a real android phone.

4.8 System Configuration Checking

System configuration is checked automatically through the android system and the application.

4.9 Tool Chain, Design Suite

What tool chain will be used? Is the tool suite up-to-date and actively supported? Is the tool suite mature enough to have stable functionality? How is the tool chain installed in a new development system.

Android Studio will be used for building the software and this is the main IDE provided by Google for android development. Therefore, the software has been up to date and should be up to date as long as android operation systems are around.

The application will be easily installed through an apk file and the stability of the application will be checked through instrumented testing that is available in the android studio.

4.10 Third Party Software

The only third party software we are currently planning to use is MpAndroidCharts. This is a third party library that is constantly maintained and all version are always available through Mavin and version control. If a future team wants to upgrade the system, then this may require some change in the code. Parts where MPAndroidCharts are used will be specifically explained in the final proposal and in the javadoc as well.

4.11 Requirements of GPR007

1.3 All software source code must be maintained under configuration control. Release snapshots must be archived on the project website.

We are using version control for the application and thus closely follow who is making changed in the software.

1.4 The system must start from cold power-up and boot to full operational status without requiring user interaction beyond enabling power and safety procedures

The application will start with user prompt and work until it is exitted(destroyed).

1.5 Any PC software must be packaged for installation with a SETUP.EXE, RPM, “make install” or equivalent installer allowing it to be installed easily on any compatible computer.

The software will be in APK format and for android environment.

1.6 Configuration parameters, calibration factors, preferences, and options shall not be hardcoded within the software source code. It shall be possible to alter these various factors without recompiling software or physically disassembling hardware. Altered configuration parameters must be persistent through power cycling and reboots. The system must have a function to initialize itself with sane (factory default) configuration content if requested.

Please see the sections, 2.6 and 2.7 in this document.

1.7 All data and configuration files must be in a generally supported format (e.g. XML) or the format required by a mature and well supported application (e.g. MySQL database files, Berkeley db, etc…). Files shall be accessible either through removable media or network file transfer or both. How are requirements in GPR007 met?

Code will be available in github, documentation will be javadoc, database system will be gson from the webserver created by SCADA team.

Document Revision History

Version Number	Approved Date	Description of Change(s)	Created/ Modified By
0.1	2/10/17	Document Created	Marty Townley
0.2	2/18/17	Document Updated	Kemal Dilsiz
0.3	2/28/17	Document updated to address comments and feedback received from the Maintainability Plan Draft (version 0.2) submitted.	Craig Lombardo Austin Wiles Kemal Dilsiz