University of Victoria
Faculty of Engineering
Summer 2015 Work Term Report

Automated Data-Driven Regression Testing for the Phone Monitoring System

CanAssist
3800 Finnerty Rd
Victoria, British Columbia, V8P 5C2

Mateus Mendonca
V00816340
Work Term 1
mfurquim@uvic.ca

May 2nd, 2015

A Work Term Report Submitted in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Software Engineering
in the Faculty of Engineering

Supervisor's Approval: To be completed by Co-op Employer

I approve the release of this report to the University of Victoria for evaluation purposes only.
The report is to be considered (**select one**): ☐ NOT CONFIDENTIAL ☐ CONFIDENTIAL

Signature: _____ Position: _____ Date: _____

Name (print): _____ E-Mail: _____ Fax #: _____

If a report is deemed CONFIDENTIAL, a non-disclosure form signed by an evaluator will be faxed to the employer. The report will be destroyed following evaluation. If the report is NOT CONFIDENTIAL, it will be returned to the student following evaluation.

Abstract

The current testing for the Phone Monitoring System takes a lot of time and effort for being necessary to trigger an event, dial a number and call in the system, and carefully perceive whether the data has changed as expected. This report is a guide through the system with software testing in mind. It will explain what is the system for, how does it work, and what should be taking into consideration when designing a test plan. This is by no means a test plan or a formal software testing documentation and it is suggested to follow the IEEE standard when writing them.

Table of Contents

Lists

i. Lists of Code Blocks

ii. List of Tables

iii. List of Images

Glossary

Although some words mentioned in the glossary slightly differ from others, they will be used interchangeable, meaning the same thing.

Artifacts and Documents	Media produced between processes to communicate efficiently.
Cron Job and Cron Task	A running process which executes something in a certain period of time.
Elderly person and person with brain injuries	The users of the Phone Monitoring System project.
Events ans Sensor's Report	Data passed from the sensors to the system meaning it changed it status. Went from deactivated to activated or vice-versa.
Features and Use Cases	Software's functions which provides useful services according to the user's requirements.
System Testing	A completely integrated system to verify the system meets the requirements.
$ symbol	When this symbol appears it means to type that in the terminal.

1. Introduction

1.1 Background

CanAssist has been developing new technologies and modifying others that already exist to help people with disabilities improve their quality of life. One of these technologies is the Phone Monitoring System, a small device capable of helping caregivers to be notified if certain events happen, such as a doctor's appointment confirmation call or a door opened in the middle of the night.

1.2 What is the Phone Monitoring System?

The Phone Monitoring System is a device that plugs to your baseline, without the use of Internet and without subscription, which helps caregivers to take care of their elderly people or people with brain injury. It records calls received, block unwanted calls and, if paired with sensors, gives a summary of what is going on in the house. If the person who is being monitored forgets something important, a doctor's appointment for instance, the caregiver will be able to listen to the appointment confirmation call and take appropriate actions. Another feature is the call out alert made by the system whenever a certain event happens, such as, opening the front door late at night or really early in the morning. Therefore, caregivers can be at ease with the Phone Monitoring System "watching" over their elderly person in these cases.

2. Features

2.1 Motives

There are a few reasons which lead to the design and development of this system. The first reason is to check out on the owner of the phone monitoring system to know if they are awaken or sleeping, if they woke up and are in the kitchen or if there is no movement in the house for a few hours. The summary is the feature to solve this need.

Another feature added in the design of the software is the call out alert, which the motive was people with dementia wandering off their houses when it is still dark at night or really early in the morning.

One more use case came from people forgetting about the calls they just received. The caregiver would come and ask if there was anything important that day, and the elderly would just answer “no”, when in fact he or she just received a call regarding a doctor's appointment for the following day. It was decided then to add the recording call feature to the system, which the test will not be covered by this report.

2.2 Summary

The summary feature comprise of the most recent activities heard from each type of sensor, door/window sensor, motion with temperature sensor, and bed sensor, as stated in the piece of code below.

```
/**
 * @brief CanAssistController::updateSystemSummaryTable
 * Summary is of each sensor type ordered by timestamp (bed sensor, motion sensor, door sensor)
 * Unless the bed is currently occupied, then only the bed sensor is reported.
 * If you add another sensor type here, make sure that the bed sensor is processed last!
 */
void CanAssistController::updateSystemSummaryTable() {
    QList<TimeSummaryPair> summaryList;
```

Code Block 2.2.1 - Update Summary

When a sensor reports motion or activation, the system filters and sort by chronological order, meaning the earliest comes first. So usually the first thing the person will hear is someone getting up the bed, then activating the motion sensor in the hall, and maybe opening the front door.

This report is going to focus on the summary use case and will recommend a few testing techniques.

2.3 Call Out Alert

This feature warns someone about a certain sensor event. For instance, the user can modify the settings in a way to enable calls from the system to his or her telephone whenever a door sensor has been activated between 8:00 pm and 6:00 am (20:00 + 10:00).

The system receives the sensor report and depending on the settings it decides whether to create or not a file with instructions so that Asterisk can make the Call. To test this feature, the directory that will contain this file must be watched with ls while fake events are fired into the system, specifying a reasonable amount of time for the system to process the event. If the file is created the test is a success, otherwise it is a failure.

There is a command line to list all the files in a directory, the well known 'ls', and if combined with another command which highlights the changes from the previous output, called 'watch', it can become a really useful tool to test this use case. Therefore, it can be used to test whether the feature is working as expected or not. The command is watch –differences ls -cr. Do man 1 watch for more information on this command.

3. Testing

3.1 Why Testing?

“When talking about risk, one has to consider the occurrence of the problem and its impacts, if it occurs” [1]. This is the importance of software testing for those systems which are considered a critical and embedded system, and have a direct impact on people lives. As said in the CanAssist website “[...] CanAssist has had a direct impact on the quality of life of many thousands of people with disabilities and their families.” [2], therefore testing this system should be taking seriously.

In the case of the Phone Monitoring System, the sensor might report to the system but if for some reason it was ignored as if it had not happened, then what? If the person with brain injury walks away at night but the door sensor does not trigger or the system does not receive it, the person might walk for hours and anything can happen until the caregiver knows about it.

Another case would be an elderly person answering the phone but for some reason the system does not record the call and it was a scam artist trying to use social engineering to acquire some information, such as the credit card number. It will be too late until someone notices that the credit card information is known by a third person and he or she is using it.

The system receiving sensors' alarms will be the focus on this report, which will explain how are the features designed, what failures might happen, how it can be tested, and how to ensure the result is as expected.

3.2 Current Testing

The current testing for the Phone Monitoring System takes a lot of time and effort for being necessary to trigger an event, dial the number and call in the system, and carefully perceive whether the date and time have changed or the summary updated as expected. So first thing to be done is trigger the event, which is opening a door with a door sensor, walk in front of the motion sensor, or sit on the couch where a bed sensor is located. After doing that, a call has to be made to the system in order to listen to the summary, or the event list, and compare with the expected result. This takes a reasonable amount of time, whereas to create an automated test would be easily scalable for a huge number of sensors and maintainable for future modifications, for instance, to add new test cases or new sensors.

3.3 Regression Testing

The intent of regression testing is to ensure that changes in the system have not introduced new faults [3]. It is executed when another feature is developed which might make references to or from any piece of code in that specific feature. Another reason for regression testing is to determine whether a change in one part of the software affects other parts as well [4], in this case the test is also ran even though it did not use any functions of that feature.

The idea of having a regression testing for the phone monitoring system is due to new features being implemented in the development stage and might be developed in the maintenance stage, such as the addition of the temperature sensors, speak through the

phone over a speaker, and black and white number lists to whether or not to record their calls, or even forbid them.

3.4 Automated Testing

Test automation is the use of another software to control the execution of tests, comparing the real outcome with the expected output [5]. While automation cannot reproduce everything that a human can do, it can be very useful for regression testing. A script to automate the test will replicate the repetitive and time consuming task of making the sensor report to the system, call in the system and listen to the summary, but it will just add fake events on the database and query another table to retrieve the readable summary, thus not having to call in the system and simplifying the process.

Automating the test will also require another program to run the script regularly, for instance, if the Linux Operating System is being used, there is the Cron program which run Cron jobs in a specified period of time. The non necessary interaction with a software tester provided by the Cron task and script driving the data to test the system will save time and turn the testers job less tedious.

Image 3.4.1 - Making a cron task running every minute

The process to make an script executable and making it run every minute is demonstrated on the image on the left. Do $ man 8 cron to more information.

3.5 Data-Driven Testing

There are a few types of test scripts, including a human behaviour script and shell script. The former is a procedure which replicates user actions, and the later is a programing code to test your application. Data-driven scripts have variables which will drive the external data supplied by the shell that invoked the test script [6]. The shell script will be used in the report to only drive data from a database as an input and expected output to test the Phone Monitoring System.

Basically, there will be a table to be read by the script test which the tests are rows and the columns are the steps on the test.

ID	1
Use Case	Alert_Sensor_Not_Responding
Pre_Test	“20 April 2015 15:00:00”
Input	“INSERT INTO sensor_data (node_id, sensor_index, command_class, name, location, measurement, value, units) VALUES ('nodeId', 'sensorIndex', 'commandClass', 'nodeName', 'nodeLocation', 'measurement', 'value', 'units')”
Pos_Test	“30 April 2015 15:00:00”
Expected Output	/* Readable time according to the function on TimeStrings.cpp (line 27) Qstring TimeStrings::getReadableTimeFromTimestamp(std::string timestamp) */

Table 3.5.1 – Explanation of data-driven testing table on database

3.6 Documentation

The artifacts generated from the testing are not the focus of this report, but if interested in documentations, the IEEE Standard for Software Test Documentation (IEEE 829-1998) [7] explains these artifacts.

4. Modules

4.1 Set Date and Time

Having a module to control the time of the system is essential for testing because the data on the table depends on the date and time the sensor's report was triggered and the current one. The readable string in the summary might be “just now” if the event was triggered a few minutes ago, or “It was opened at” if it has been more than 5 minutes. Depending on the sensor's type it may also say “It was triggered” or “It has been opened since”.

This module will control the date and time before and after faking the event into the database, and even after modifying the pos-test date, it has to wait something around one minute and a few seconds to the system to update the table. After giving the program this moment to process the table's information, the query will retrieve the data from it and a Regex rule will verify the accuracy of the outcome. Do '$ man 1 date' for more information about setting the system's date and time.

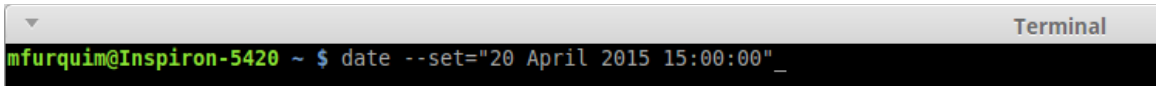

Image 4.1.1 – Modifying system's date and time.

4.2 Fake Sensor's Report

There are two ways to fake the sensor's report into the system. Either sending the fake report to the ZStick located at `/dev/ttyUSB0`, or writing directly on the database. Faking an array of bytes to try to replicate the sensor's report requires an understanding of what each byte means, whether is the time stamp, temperature, sensor activation, or

whatever the sensor reports. Whereas writing in the database is already in a fairly understandable format, thus it will be used to fake the events.

Although there are a few tables in the database to be concerned about when designing the test plan, only one will be used to write the input. The tables are events. sensor_data, summary, and alerts table and their schema is as described in the picture below.

```
if( mysql_query (connection, "CREATE TABLE IF NOT EXISTS sensor_data (node_id NUMERIC, sensor_index NUMERIC,
command_class NUMERIC, PRIMARY KEY(node_id, sensor_index, command_class), name TEXT, location TEXT, measurement TEXT,
value TEXT, units TEXT, update_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP)")) {
    Log::Write(LogLevel_Warning, "Could not create the sensor_data table\n%s", mysql_error(connection));
}

if (mysql_query (connection, "CREATE TABLE IF NOT EXISTS alerts (alert_id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
type NUMERIC, summary TEXT, description TEXT, readable_time TEXT, timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY(alert_id))")) {
    Log::Write(LogLevel_Warning, "Could not create the alerts table\n%s", mysql_error(connection));
}

if (mysql_query (connection, "CREATE TABLE IF NOT EXISTS events (event_id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
node_id NUMERIC NOT NULL, sensor_index NUMERIC NOT NULL, command_class NUMERIC NOT NULL, measurement TEXT, monitor_id
NUMERIC NOT NULL, summary TEXT, description TEXT, readable_time TEXT, timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY(event_id))")) {
    Log::Write(LogLevel_Warning, "Could not create the events table\n%s", mysql_error(connection));
}

if (mysql_query (connection, "CREATE TABLE IF NOT EXISTS system_summary (summary_id BIGINT UNSIGNED NOT NULL
AUTO_INCREMENT, summary TEXT, timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP, PRIMARY KEY(summary_id))")) {
    Log::Write(LogLevel_Warning, "Could not create the system_summary table\n%s", mysql_error(connection));
}

if (mysql_query(connection, "CREATE TABLE IF NOT EXISTS sensor_activity (node_id NUMERIC, name TEXT, location TEXT,
last_active TIMESTAMP DEFAULT CURRENT_TIMESTAMP, PRIMARY KEY(node_id))")) {
    Log::Write(LogLevel_Warning, "Could not create the sensor_activity table: %s", mysql_error(connection));
}
```

Code Block 4.2.1 – Tables schema, lines 236 to 265 on the CanAssistController.cpp

The table sensor_data contains information about the sensor, such as, name, location, measurements, and value. Those values should be updated in order to trick the system to believe the real sensor has been activated.

```
QString sqlInsert = "INSERT INTO sensor_data (node_id, sensor_index, command_class, name, location,
measurement, value, units) VALUES(" +
        nodeId + ", " + sensorIndex + ", " + commandClass + ", '" +
        nodeName + "', '" + nodeLocation + "', '" + measurement + "', '" + value + "', '" + units + "')" +
        " ON DUPLICATE KEY UPDATE Name=VALUES(name), Location=VALUES(location),
        Measurement=VALUES(measurement), Value=VALUES(value), Units=VALUES(units),
        update_time=CURRENT_TIMESTAMP";
```

Code Block 4.2.2 – Inserting value into the input table

The following table is an example of selected columns from the table's schema after the fake event.

Measurements	Summary	Readable_time	Time_stamp
Sensor	The sensor in the Kitchen is activated	At 2:35pm on April 24	2015-04-24 14:35:51

Table 4.2.3 – Examples of selected important columns from the events table

4.3 Query the Output

Roughly one minute has to be given to the system to process the information after setting date and time, fake the event into the database, and setting the date and time after the insertion. To ensure the outcome of the whole process is as expected, first the row has to be fetched from the database using the “SELECT” operation, after that the “summary” field has to be compared with the location, the “readable_time” has to be compared depending on what is the type of the sensor and the time it has been since it was activated. Besides the events table, there is also other two important table that worth querying to check the outcome: summary and alerts.

4.4 Regex

Using regular expressions might be the easiest way to compare words and find the string that will make your test pass. Depending on what is being tested, setting the location on the sensor_data for instance, the regex will try to find that word on the

summary field, and if it can find the correct name, the test is a success. Another thing to be tested are the readable time, but it depends on the type of sensor and time the fake event was triggered.

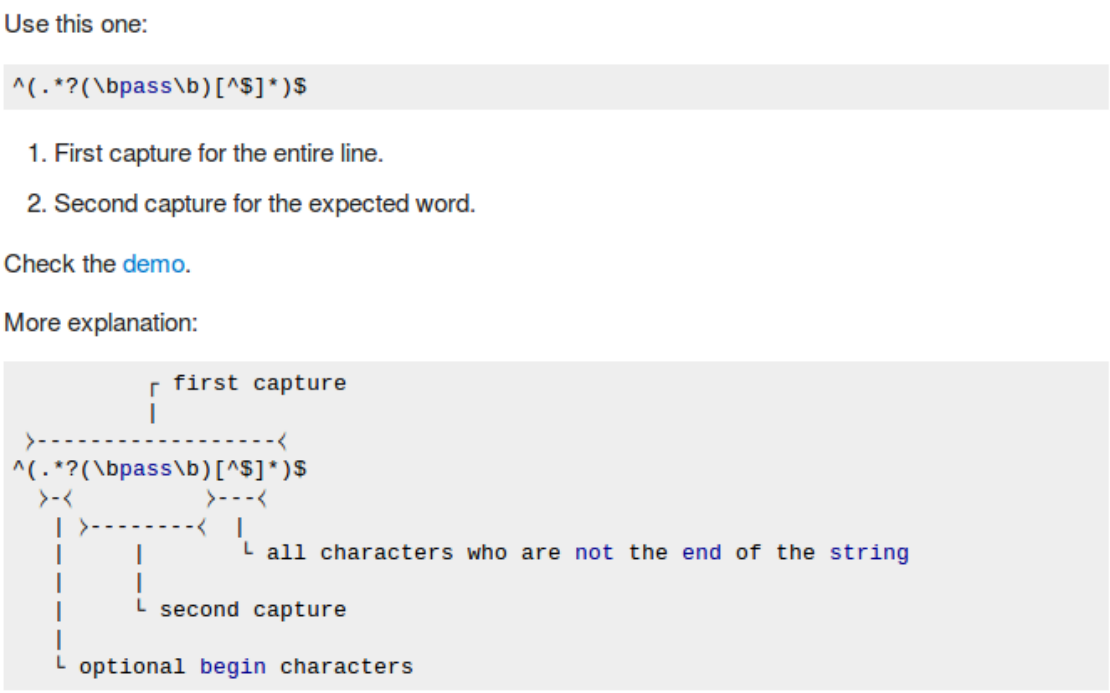

Image 4.4.1 – How to capture a certain word in regex. [8]

4.5 Use Case Selector

The last module needed is a use case selector in case more than one feature want to be tested. It will use the second field, named use case, from the test table to switch between the tests created. Each test in the script is a function and will drive the data differently. This differentiation will make it easier to add new testing features by just adding one more routine for each new use case to be created and change the use case field in the table.

5. Conclusion

5.1 Comparison

Comparing the current test and the automate data-driven regression testing, is easy to see how an automated test will help in the future development and maintenance of the Phone Monitoring System. By spending time and effort to develop the proposed table driven by the testing script, with modules well defined to be specialist in doing an specific task, the ensured quality of the software will be higher than having a tester to trigger an event and check manually whether the result is the expected one.

6. Bibliography

[1] Soares G., “Software Testing and its Relationship to the Context of the Product” [Cited May 2nd] (translated from Spanish), October 2011.

[2] CanAssist, “About us”, [Cited May 2nd 2015], 2015, available at: http://www.canassist.ca/EN/main/about-us.html

[3] Myers, G. J., 1946, Sandler, C., 1950, Badgett, T., & Ebooks Corporation. (2012). The art of software testing. Hoboken, N.J: John Wiley & Sons.

[4] Savenkov, R.. “How to Become a Software Tester” (2008), pp. 386.

[5] Kolawa, A., Huizinga, D. (2007), “Automated Defect Prevention: Best Practices in Software Management”, pp. 74.

[6] Carl N., Data Drinve Automation Frameworks, [Cited May 3rd, 2015], available at: http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm

[7] IEEE, 829-1998 Standard, [Cited May 3rd, 2015], 1998, available at: http://standards.ieee.org/findstds/standard/829-1998.html

[8] StackOverflow, “regex find word in the string”, [Cited May 5th, 2012], February 2012, available at http://stackoverflow.com/questions/9348326/regex-find-word-in-the-string