Software Testing Plan

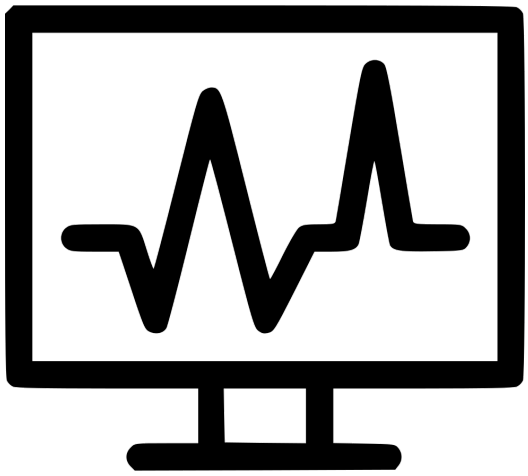
4/6/2018

Dr. Fatemeh Afghah

Version 1.0

Nathan Payton-McCauslin, Alexander Grzesiak,

James Todd



Data Menders

Table of Contents

1.0 Introduction

This document serves to outline our team's testing strategies for our project. This includes testing individual modules and components as well as testing the interactions between them and testing for application usability. The overall goal of our application is to detect different types of alarms and differentiate which are truly true and which are meaningless (false). The success of this project lies in the accuracy we can obtain for alarm differentiation and thus requires rigorous testing.

Software testing can be broken up into three primary components: unit testing, integration testing and usability testing. Unit testing tests the core of the software application, in particular, it tests that the code is actually working as intended in the backend. Integration testing tests that the interactions between different modules and pieces of code are acting as we expect. Usability testing tests how easy the software is to use for the target audience, ie. how intuitive it is to use. Each if these have important roles and cannot be ignored, unit tests are a staple of software development since working code is the backbone of any software while integration testing becomes more important when there are many sources of smaller pieces of software such as our project. Usability testing is mainly used in User Interfaces (if the software has one), in our case, this is all about our Graphical User Interface since the user will only interact with that.

For our software application we use all three of the above methods of testing to varying degrees along with some other methods that do not fall into one of those categories. However, unit testing for example, will not be done for code that has been given to us by our mentor and affiliates or built in or imported libraries because it is assumed that this code works correctly. What we will be unit testing are pieces of code written by our team, both utility and driver code that intertwines with given code and built in and imported libraries. Integration testing will make up the bulk of our testing due to the large amount of pre-existing code and libraries that are code works with. The two biggest pieces of integration testing will be making sure that the backend code communicates correctly with the front end GUI and that our code is properly configured to work correctly with the machine learning package we are using (Weka). Our usability testing consists of making sure the interface is easy to use and intuitive. In order to do this,

usability testing has been broken into two phases: internal testing and user targeted testing. Internal testing is simply among our team and mentor, making tweaks as we see potential usability problems. User targeted testing is much larger and more important, this testing has to do with getting a sample size of intended users to actually use our application and provide us with feedback as necessary. As mentioned above, not all testing we are doing fall under unit, integration or usability. Other testing methods we will use are manual testing and using the Weka GUI to test for accuracy within alarm differentiation. Manual testing is detailed in (enter section number here) and includes types of testing that cannot be described by the standard three types.

2.0 Unit Testing

For unit testing within our application we will be using the xUnit Matlab library because this allows us to mock up test situations, easily run multiple tests at once and it shows code coverage. We we will only be unit testing a subset of the code in our application due to a large part of our code base being code from other places such as, internal and external libraries and code built by others working on similar projects here at Northern Arizona University.

In Matlab you can create seperate functions called test functions to allow testing of another function using mocking of the other function parameters. Mocking refers to using data that we know should yield a certain result and the test tests that it actually gives this result. An example unit test is shown below where the function to test just finds the length of a string and the test function is a unit test that tests the result is what we expect.

Example function to test:

```
function size = length(string)

end
```

Test function

```
function testLength
        word = ‘me’;
        len = length(word);
        if norm(len) ~= 2
                error ‘Test Failed’;
        end
end
```

The reason we will be using the xUnit library is because it will run all functions in the current directory whose name starts or ends with test or Test. In addition to this, it will also tell you

which passes out of those and which failed as well as the time taken to complete the tests (this allows to also test efficiency at the same time).

3.0 Accuracy Testing

3.1 Weka Testing

While we are using Weka inside our actually application in the backend Weka also provides a GUI that we can use to quickly run extracted feature classification to determine the accuracy of those features. The accuracy this refers to is defined by how well Weka is able to classify alarms (True or False) based on this feature set. We are able to check this because we have a training set of patients where we are provided the alarm classification for each patient. By extracting these alarm classifications as another feature, we can run this with our other features we extracted and determine how often it actually predicts the alarm classification correctly.

There are many combinations of classification techniques in Weka and each provide us with different accuracy levels for each set of features, we need to discover the best combination to use for our extracted features. Since we use embedded Weka code in our Matlab application to run through all these features and perform selection and classification, this part is necessary to obtain accurate features before we do all the computation.

3.2 Comparison Testing

Comparison Testing

Since we are not the first to try and solve the over saturation of false alarms in intensive care units we are able to manually compare our results with the results of other software designed to also decrease false alarms. This is important in order to show the world that our project is better and worth while. In addition to showing that our project is more accurate at decreasing false alarm rates in ICUs, we will also be able to show through testing that our project is cheaper/cleaner/more friendly than other projects. The plan here is to make adjustments to our project based on things that other projects do not get right, at the end our project should do all the things right that the competition does wrong.

4.0 Integration Testing

Since our project has many modules working together, it is the goal of integration testing to ensure that all the modules in our application work well together. Each subsystem of our application as a whole has one job but the systems don't talk to each other in the way that processes might send messages back and forth to each other. In our case the communication that each has with the others is in the form of files containing patient data, signal processing outputs, or log files which illustrate the process by which some files were generated. Even with most of the information being passed between processes with files, there are still instances where information was passed around my parameters. These two cases were the only ways information was passed around the different parts of our application and as such there are only few ways we tested them.

The first method we tested was the generation of files at different points in the execution of the program. The flow of the data transformation process starts with a set of raw signals. These raw signals contain ECG, ABP, and PLETH signals. The ABP and PLETH are signal signals and are unique to their names. ECG on the other hand can be of three different types. It is possible to have ECG I, II, and V which are different enough that we need to handle them accordingly. When signal importing is being done the signals are designated according to their type and this testing was done by asserting that the information that was being imported was the same as what we expected when looking at the data. This data was then written to a file; the raw signals can be assured to have integrity.

After that the flow takes the raw signals to the signal processing step where signal transformations are performed to get many new signals from one raw one. The methods we used here are either built into Matlab or are proprietary to NAU's SICCS program so these were taken at face value. Testing with these was essentially calling the function and asserting that the value that came out was the same as expected when referencing the Matlab documentation. Once the raw signals were processed, they were saved again as processed signals. Again, the data being written to file is accurate and in a format that was set forth in our planning. The format was tested by opening the files and asserting that the values in them were consistent with known processed signals.

After the signals have been processed and there are now two types of files, 3 raw signals for each patient and a multitude of processed signals for each of the 3 raw signals. Now comes feature selection which will take each processed signal and extract useful information from those signals. Each processed signal ranges from having only 1 row of data to having hundreds. So in order to test potentially large processed signals, we asserted that the files being produced were consistent with what we calculated manually.

After the processed have had useful information extracted, classification uses the files that contain the information to decide which pieces of information are the most relevant. This part includes opening a file with the information and passing it around a framework to do the classification. The framework uses machine learning algorithms which have been well documented as accurate so this aspect is taken on the

behalf of Weka to be working properly. The output from this section are trained models that can be used with alarm determination which is the end of the flow of testing for this section.

When it comes to how information is passed around functions within our application, the parameters are generally limited to file names or constants such as frequency and points in time. Parameters such as file names can be tested by asserting the names gathered by looking in a specific directory are in fact the files that exist in said directory. Parameters such as frequency can be tested by asserting that the frequency being passed around is the same that is present in the header files of the raw signals.

Finally, returned values are rare given that most of the information that needs to be returned is exported to files for the sake of memory conservation. However, there are cases where functions are returning some sort of data. The functions that do return values are often using built in operations or are trivial by nature but these functions are tested by asserting that the values coming back from the functions are what we would expect and are in the proper format.

5.0 Usability Testing

Since the final product of this project is a graphical user interface, usability testing is highly important. There will be five main parts to the graphical user interface, the Home screen, the signal processing screen, the feature extraction screen, the feature selection screen, and the signal test screen. Each screen will need to have a form of usability testing done to insure that the end user will have a smooth learning experience using this product.

5.1 Usability Testing for the Home Screen

Usability testing for the Home Screen will not be that complex. The first time the home screen runs, the user will need to be connected to WiFi or to the internet. This is because the Home screen on the first time running a setup command will run and will set up a Target folder. The Target is a file structure that will have folders for each screen and a preprocessing data folder. The preprocessing data folder acts as a cache. Also within Target, a folder will be downloaded, this folder will be the training set. The preprocess data folder will be filled with data from the training folder. Also the only functionality within the Home screen that the user interacts with are buttons. These buttons will open the other screens talked above in section 4.0. Some tests will be:

1. Check to see if that Target exists. If not, make it.
2. Within Target, check to see if the training folder exists.
3. Check to see if the preprocessed data exists.
4. Test that the signal processing button opens the signal processing screen.
5. Test that the feature extraction button opens the feature extraction screen.
6. Test that the feature selection button opens the feature selection screen.
7. Test that the signal test button opens the signal test screen.

5.2 Usability Testing for the Signal Processing Screen

Usability testing for the Signal Processing screen will have more challenges than the Home screen usability testing. Signal Processing screen will have more user interaction than Home screen does. Each possible user interaction within this screen is statically set. This is so that users can only pick what is there and not input what they want, this will make testing things easier. Signal Processing will get its data from within Target in preprocess data. There are six different kinds of signal processing transform that a user can pick, also there are three different types of signal a user can pick. Again, because these selections are static, the user only needs to pick what they want, removing the need to type input from the user. Something to keep in mind is that a signal process transform, H.R.V, can only be used on one of the signals types. So this option needs to be removed when the wrong signal type is picked and it also needs to be added when the right signal type is selected. From there, when the user picks all of their selections,

then the user can hit the Start button. Once the start button is selected, it will need to be disabled. This is so that the user does not hit the start button twice, this could be bad because it could start two processes of the same thing, which is redundant. Some tests are:

1. Make sure that the user selected something, do not run on no selections.
2. Test to see if the Start button is turned off after being pressed
3. Test to see if the Start button turns back on after signal processing is finished.
4. Test to see if the right data is being loaded.
5. Test to see if H.R.V is turned off and set to No if the right signal is not selected.
6. Test to see if the selected transforms are the ones that are ran.
7. Test to see if the processed data is save in the right spot.

5.3 Usability Testing for the Feature Extraction Screen

Usability Testing for Feature Extraction will almost be identical to Signal Processing because the layout of the graphical user interface is almost the same. The difference is the names. The inner working of the graphical user interface is a little different because different processing needs to take place. Each signal that was processed within Signal Processing will have Features that need to be extracted. So the data will need to be taken from the signal processing folder within Target. Then once the data is processed, save it with the feature extraction folder in Target. Some tests are:

1. Make sure that the user selected something, do not run on no selections.
2. Test to see if the Start button is turned off after being pressed
3. Test to see if the Start button turns back on after signal processing is finished.
4. Test to see if H.R.V is turned off and set to No if the right signal is not selected.
5. Make sure the data loaded is from Signal Processing folder, and from the selected transform(s) and signal type.
6. Test to see if the selected transforms are the ones that are ran.
7. Test to see if the processed data is save in the right spot.

5.4 Usability Testing for the Feature Selection Screen

This Section is just like section 4.3. However, what is changing is now the extracted features need to be put into a program called Weka so that the important features are selected from the extracted features. The graphical user interface is just like the ones in sections 4.3 and 4.2. The data for this will be taken from the Feature Extraction Folder, and saved within the Feature Selection folder. Some tests are:

1. Make sure that the user selected something, do not run on no selections.
2. Test to see if the Start button is turned off after being pressed
3. Test to see if the Start button turns back on after signal processing is finished.
4. Test to see if H.R.V is turned off and set to No if the right signal is not selected.
5. Make sure the data loaded is from Feature Extraction folder, and from the selected transform(s) and signal type.
6. Test to see if the selected transforms are the ones that are ran.
7. Test to see if the processed data is save in the right spot.

5.5 Usability Testing for the Signal Test Screen

The Signal Test screen will be one of the harder ones usability testing on. Here, this screen can load in a user selected patient from a static list from the training folder within Target. This patient will be loaded in and processed by all of the possible signal transforms, feature extraction, and feature selection will be done on them. The tricky thing here is to see if the selected patient has the signal types we need. At most, one is needed, but the more the better. Once that is all done, each selected features file from that patient will need to be tested against the training set that was made from the other parts of this program. From there it can be determined if that patient's alarm is false or true. Also, there will be a graphical functionality of the graphical user interface where the signal from the selected patient is "read in" as online data. Some other stats of the signal will also be displayed about the patient. This "read in" process will be a little challenging as well because the goal will be to treat an offline signal as an online signal. So windowing of the signal will need to take place to treat it as an online signal, online can be seen as live input. There will also need to be a way to input a signal that is not from the static set, this will most likely be another folder within the Target that will hold these files. These files will only be ECG signals. These file will follow the same flow as the other static set from the training set. Some tests are:

1. Test that the patient data that is loaded is correct
2. Test that all processing on this patient is done
3. Test to see that a costume ECG data can be loaded.
4. Test to see that all processing on this costume ECG is done.
5. Ensure that the graph is updating correctly
6. Ensure that the stats are being displayed
7. Ensure that the program is telling the user what is going on
8. Ensure that the outcome and expected outcome is displaced