Northern Arizona University

Team Radio Pi

Capstone Team	*Project Sponsor*	*Mentor*
Tyler Plihcik	**General Dynamics**	David Failing
Brandon Click	**Mission Systems**	
Jefferey Williamson	Randy Derr	
Kaelen Carling		
Saurabh Jena		

Software Testing Document

Version 1.0

March 8th, 2021

Table of Contents

Introduction

The world runs on multitudes of IoT (Internet of Things) devices; everything from smart home devices, to wifi extenders, speakers, even cars and kitchen appliances. These devices communicate via radio waves and run on what are called embedded chipsets, which are inherently small, in both footprint and power consumption. As our technology continues to advance, the need for smaller and smaller embedded systems becomes more urgent.

For nearly three decades software defined radios (SDRs) have been revolutionizing the way these IoT devices communicate. Leading the charge into this new era of is our project sponsor, General Dynamics Mission Systems. The need for newer, smaller, more efficient yet more powerful systems is great. Current solutions are often times built on legacy software, and although they still work, are definitively not the best solution given our technological advances since the late twentieth century. Because of this, Software Defined Radios are prone to:

- Failure
- System crashes
- Memory overloads
- General decay

Our envisioned solution is to redesign General Dynamics SDR Command and Control System from the ground up. We will be simulating this solution on a Raspberry Pi, a very small yet powerful, computer. We will establish an interconnected database and web interface in the most efficient language(s) available, including networking protocols and database languages. We will then ensure that the end user has complete and total control, given they have the proper authorization, over the system. From querying the database to collecting, sorting, and storing data to controlling permissions and viewing system diagnostics, our envisioned solution will attempt to right all the wrongs of current command and control interfaces for embedded systems.

The following will serve as the software testing plan for our solution, detailing the testing process and project testing benchmarks. Our solution can be easily broken up into three major parts: front end user interface, our custom shepherd system middleware, and the backend database and networking functionality. More focus will be on individual unit testing than on integration testing because we have so many more units than whole pieces of the project. Additionally, one of our clients major concerns is how our solution will handle under load, so a large portion of the unit testing for our backend, namely, our database will focus on stress testing and security testing for our networking functionality as well as our database administration actions.

Unit Testing

Software unit testing is the first stage of testing in the software design process and life cycle. It involves taking apart each of the major portions of the project as a whole and breaking them down into their individual components. For example, a car as a whole can be broken up into major components: engine, chassis, and enclosure, but you can further break those down into the engine block, the timing belt, then fuel injection system, the wheel base, suspension, doors etc… The main point of this unit testing is to ensure that everything functions properly on its own before it is integrated into the system at large. This guarantees no major failures when everything is being put together.

Front End User Interface

Our front end user interface (UI) will be the face of our project, what the client sees in every demo, and what they will see and interact with every time they use our solution. The front end is built using basic HTML in combination with Flask. It will consist of the actual functionality of the HTML, ie the forms, submit buttons, and responsiveness, and the functionality of our flask code, which will be making the connections to our backend, this is the actual logic that begins as soon as the user presses the submit button on any data entry field.

In order to test the HTML portion of our front end, we need to ensure the following:

- That all forms are set up properly with the correct method (POST, GET, etc..)
- That all forms are configured to transfer to the correct information to the correct destination, ie the action
- That all submit buttons have the proper value on them
- That all forms intake the proper input type (number, text, etc..)

To test that all forms are set up properly for our UI, (this includes the first three bullet points from above) we will perform manual testing on each form with a random set of acceptable values, and confirm receipt of those values by observing the actions performed by the front end after their input and submission. Proper set up for the forms that make up our UI will be confirmed if upon submission of our random sets of acceptable values the front end successfully has sent the information entered in the data entry field(s) from the UI, however receipt of these values will be detailed and tested later, for the purpose of this particular unit test, we simply want to confirm the send.

To test that all forms intake the proper input type we will perform manual testing on each form by inputting a random set of both acceptable and unacceptable values, submitting them, and observing the result. Proper intake will be confirmed if there is no error message telling the user that their input was not valid.

In order to test the Flask portion of our front end, we need to ensure the following:

- That receipt of the values in their proper destination is confirmed
- That when the submit buttons are pressed, the proper database queries are sent
- That when the proper database query has been sent, the proper database response is sent back
- That the front end itself is actually connected to our networking component via sockets

To test that the values are being sent to their destinations, after inputting a random set of acceptable values we will observe which actions of the flask code are triggered, ie if we enter a lookup for the device information of device identification number three, that the primary key deviceID with the value 3 is sent to the proper flask function which then triggers our next test.

Next, to test that the proper database queries are sent and correct information from the database is received, after successful submission of acceptable values into our data entry fields, we will record the database query that is sent to our database, and observe the value that is sent back to the UI. If the information sent back to the UI in the use case detailed previously matches the database records for that specific device then the test has succeeded.

Finally, to test that the front end itself is actually connected to our networking component via sockets we will simply try and query anything particular piece of information from any data entry field using a random set of acceptable values. Any response containing information from our database will confirm successful connection.

Database

The integrity of our database and its proper function are essential to the success of this solution. In a way, the database is the heart of our project, and its testing is of paramount importance. The database holds vital information such as device identification numbers, core computer data and log files etc…

In order to test the database, we need to ensure the following:

- That the database continues to perform under load
- That the database size remains small
- That the database connection persists throughout a user session
- That the database responds to queries accurately and in a timely manner
- That the database is secure and not subject to attacks such as SQL injection attacks

To test that the database performs under load we have performed a great deal of stress testing. This was one of the client's biggest concerns. We will perform various tests in an attempt to answer the following questions:

- How much space does each record occupy in the database?
- How fast can the Radio Pi write to the database?

In order to do this, due to the nature of stress testing a database we will create our own automated test functions that will run at the Pi's maximum computing speed. We will then record and document the results.

To test that the size of the database remains small we will perform a test in which we run the Radio Pi nonstop for an arbitrary period of time where it writes to the database. We will end this test when the database breaks due to size, or we reach a database size exceeding client expectations while the number of writes to the database falls below client expectations.

To test that the connection persists throughout an entire user session we will perform a test in which we write to the database an arbitrary number of times and ensure that the data sent is received and placed in the database.

To test that the database responds to queries accurately and in a timely manner we will write a test function that queries the database, and records the time that the query is sent. Upon the query response we will record the time in which we get it and that the received response is the correct response.

To test that the database is not subject to security vulnerabilities such as SQL injection attacks we will attempt our own SQL injection attacks against the database, performing our own penetration testing. Success for this depends on whether or not we are able to successfully attack the database.

Networking

The bridge connecting our major pieces is just as important as the pieces themselves.

In order to test the networking component of our solution works properly, we need to ensure the following:

- That the connection between the front and back end successfully transmits data to the correct destinations

To test that the connection between the front and back end of our solution has been successfully established we will configure messages that are sent and visible if and only if networking component is fully operational and functioning without error. If we do not receive these messages upon establishing connection, the test has failed and the networking component is not functioning.

Integration Testing

After a successful round of unit testing, the next step within the software testing process is integration testing. Integration testing is the practice of taking the individually tested units and beginning to piece them together one by one. The goal of this testing is to ensure that the major parts of the project work well when put together, because although specific parts may work perfectly on their own, in a final solution to an issue all parts need to work together as a whole.

Our approach for integration testing will come in three major parts:

- Our network integration to the database
- Our network integration to the front end
- Our database integration to the front end

Simply put, we are establishing the bridge from either side to the middle, and then driving our car across it to ensure full product functionality.

Network Integration to The Database

The first step in our integration testing will be confirming that our database can establish a successful connection to our networking component.

In order to test the integration of our networking component to our database we first we will ensure that all ports are configured to always listen. Doing this will confirm any issues encountered will be with the network connection itself, rather than the database config. Once a connection is established between the database and the networking component, we will see a series of automated messages that get pushed to the user / tester in the event of a successful connection.

After confirming a successful connection of the network to the database, we will disconnect from the database, and again, a series of automated messages will be pushed to the user / tester in the event of a successful disconnect. All other outcomes of this test will result in a blatant error message thrown to the user / tester.

Network Integration to The Front End

The next step in our integration testing will be to confirm that the network is connected to the front end user interface.

To test that our network has successfully integrated to the front end UI, we will have a message set up to automatically send to the user interface upon a successful connection establishment. Receipt of this message will confirm that the front end and the network are connected and talking to each other, any other result will be considered a test failure.

Database - Front End Integration

The piece of the puzzle that truly makes our solution function will be the successful integration of our database to our front end user interface. The database is so essential because it holds and provides all of the information on the embedded device(s) that the user will need to view.

In order to test that our database and front end have been integrated properly we must ensure the following:

- That the front end can ping the database over our network, confirming connection between the two
- That the front end can successfully submit queries to the database based on user action
- That the database can successfully send information based on received queries back to the front end

To test that our front end can ping the database over our network, which will confirm connection, we will simply ping the database and wait for a response back. If we receive no response back then we no that the connection is no longer functioning.

To test the final two points from above, we will write tests that send multitudes of arbitrary queries to the database, log the responses we get, and then confirm that the information received was the information expected. Any information received that was not expected based on the sent queries will be considered a test failure.

Usability Testing

Usability testing is an often overlooked part of software testing. Whereas the above tests, unit and integration beg the question “did we build the product right?”, usability testing asks “did we build the right product?”. No solution, no matter how well it functions or how slick the code behind the scenes is or how fast the algorithm is will be considered successful unless the users can operate it easily and intuitively.

Our user interface is made up of three primary sections: the device ID request section, the device activate / deactivate section, and the status view for all devices.

Because our solution is covered under a non-disclosure agreement between the team and General Dynamics Mission Systems, normal usability testing is impossible for us to perform. However we will measure usability internally through time standards. That is, how fast it takes a user to perform a certain action. Obviously we must take into account that our team will be very familiar with the system, to reflect this in our testing we will use significantly lower time standards with ourselves than we would if we were testing with a true user.

The device ID request section works as follows, the user has successfully navigated to the dashboard and locates the device ID request section, they then enter a valid device ID and click the button below the entry field and results are displayed. We have determined that the internal test user must be able to complete this action in under five seconds to be able to give this section a usability testing pass.

The device activate / deactivate section works as follows, the user has successfully navigated to the dashboard and locates the device activate / deactivate, they then enter a new device ID and click the activate button. Following this action they will enter the same device ID in the deactivate section and click the deactivate button.We have determined that the internal test user must be able to complete these actions in under eight seconds to be able to give this section a usability testing pass.

The device status view for all devices works as follows, the user has successfully navigated to the dashboard and locates the device status view section for all devices. Since the page has just loaded there is no need to refresh yet. The user will be able to view all devices that have been registered with the Radio Pi Command and Control System accompanied with their status. We have determined that the internal test user must be able to complete this action in under three seconds to be able to give this section a usability testing pass.

Conclusion

A full testing regime is a vital part of any project, no matter how big or small. The combination of unit testing, integration testing, and usability testing in that order will provide the development team with the most complete and compliant solution ready for submission to the client.

This document has served as our “blueprint” for our software testing plan, and based off of the standards set in place here, we will be testing our solution for General Dynamics to the strictest standards to ensure that the solution we deliver is absolutely complaint to documented requirements, and that it exceeds client expectations.

Glossary

UI - user interface, the interaction point between human user and computer system

HTML - Hyper Text Markup Language, the markup syntax used to write web pages

Flask - a python micro framework used in web service development to bridge the gap between front and back end