Test Document

The Disease Outbreaks Team

Abdulaziz Alhawas

Jean-Paul Labadie

Jordan Marshall

Luis Valenzuela

CS 486C

4/14/2016

Dr. Wolf-Dieter Otte

Introduction

Testing is an important and vital part of any system’s development lifecycle. This testing plan has been created to help communicate the the approach that our capstone team has taken and will take to ensure the fidelity of the end product. In our project, we used various methods for testing our code and components to secure that everything is running smoothly and working as intended. The main sections of this document will focus on unit testing, integration testing, and usability testing as the methods we have leveraged for our project.

Unit testing was used on our source code and test cases were created to make sure that the methods are doing what they have been designed to do. After unit testing, integration testing was used not only to ensure that our product works extremely well within its own instance that affords local usage, but also with Tgen’s computing cluster to make sure that no exceptions occur when interacting with Tgen’s servers.

Since our project is a user interface based system, usability testing is important to the success of the product by testing with real users. We had meeting with Tgen’s employees that were going to use our product and we took time to understand their trends and what they like or dislike about interfaces that they currently use at their workplace. This helped to build an intuitive and highly workable interface that our audience will be happy to use. This document will clearly outline the process and approach of our testing plans that came together to create the end product.

Unit Testing

The graphical user interface being implemented consists of a variety of parts that all interact with each other to provide useful functionality within the interface. Many of the modules that provide that functionality depend on the correctness of other modules. Because of this it is important to make sure that they are functioning as expecting and returning values that make sense and are correct for any given input.

Developing unit tests will ensure that our modules are correct and will help us identify and correct issues within the program. The following are important modules, their arguments, and a discussion of the unit tests that will validate their return values.

Job Saving/Loading Module

This module provides a time saving feature in the form of loading and saving job settings. It saves settings by taking the contents of the interface and putting them into a Java object named NaspInputData. The contents of this object are then saved into an XML file that is formatted in such a way that it can be sent and run on NASP without having to be processed further. The loading works in the opposite direction. First an instance of the NaspInputData object is created, the contents of the XML file are then saved into this object, and finally the interface is populated.
Methods

Methods and Unit Tests

jaxbXMLToObject

This method takes an XML file as argument and returns an instance of NaspInputData that contains all of the settings within the XML file.

jaxbXMLToObjectTest

This unit test passes an XML file to the jaxbXMLToObject method and stores the reference to the NaspInputData class that it returns. The values within this class are then compared to the corresponding values within the XML file. This test passes if the values match and fails otherwise.

User Settings

The user settings module handles the saving and modifying of settings such as username, URL, port, and job manager. These settings are stored on initial login and later used to connect to a given cluster therefore it is important to make sure that the information is being stored and returns all of the fields correctly

Methods and Unit Tests

getCurrentRemoteSettings

This methods loads the current remote settings from the configs directly into a JSON object. It then accesses a key within this object corresponding to the remote settings and returns its contents in the form of a JSON object if the key exists otherwise it returns null.

getCurrentRemoteSettingsTest

The test for this will call getCurrentRemoteSettings, save the JSON object returned, and validate that the contents of the object to ensure that the returned objects contains the remote settings.

setCurrentRemote

This method simply sets a new setting with the key “Current Remote”. Initially the contents of this key are empty.

setCurrentRemoteTest

The test for this method simply calls the setCurrentRemote then calls getCurrentRemoteSettings and ensure that there actually is a “Current Remote” key in the JSON file.

addRemoteSettings

This method actually sets the contents of a setting key within the JSON configuration file such as the Current Remote key.

addRemoteSettingsTest

Again this test will call the method to add the remote settings to a key and then verify that the contents of that key are correctly set.

removeRemoteSettings

This method takes the name of a setting as an argument. It then uses this name to search the JSON configuration object and remove the corresponding settings.

removeRemoteSettingsTest

This test calls the method on a JSON file and then makes sure that the setting is removed by trying get the contents of the key removed. This should return null.

readSettings

This method takes the path to a file as an argument and returns a JSON file with the contents of that file.

readSettingsTest

This test calls readSettings and compares the contents of the returned JSON object with another JSON object that should match.

writeSettings

This method takes a path and a JSON object as arguments. It then writes the content of that JSON object into the path specified.

writeSettingsTest

The test for this calls writeSettings and then calls readSettings and verifies that the contents of the JSON file are correct.

getCurrentServerURL

This method loads the current remote settings into a JSON object and returns the contents of the URL key.

getCurrentServerURLTest

This test will call the method and verify that the URL returned matches the expected URL.

Network Module

The network module handles all of the communication with the clusters. This modules role within the overall program is important as its the bridge between the interface and the NASP tool. It needs to be able to form a connection, send commands, and transfer files from the cluster to the local machine and vice versa. The following tests will ensure that the methods that this module is comprised of function correctly.

Methods and Unit Tests

intiSession

This method takes username, password, url, and port as arguments. It then simply creates an instance of the Session class.

initSessionTest

The test for this calls initSession on an instance of the NetworkManager class, calls the initSession method of the class, then calls the getSession method, and checks if the returned value is null or not. If it is null the test fails.

openSession

This method uses the session instance that initSession creates to form a connection using the arguments passed to in.

openSessionTest

This test passes if the call to openSession does not result in an exception. Open session will attempt to create a connection will throw an exception is it fails.

closeSession

This method closes all channels that are currently open and terminates the connection.

closeSessionTest

This test calls closeSession on an open session then calls isConnected on the session. If this returns null the test passes.

upload

The upload method takes a file and an absolute path as arguments. It then uses the connection to the cluster to transfer the file to the cluster into the specified path.

uploadTest

This test calls upload and then checks for the file in the directory on the cluster. It passes if the file exists.

download

This method takes the an absolute local path and an absolute remote path as arguments. It then copies the file from the absolute remote path to the absolute local path.

downloadTest

The test calls download and checks whether the specified filename exists in the local directory.

runNaspJob

This method takes a path to an XML file as argument. It then uses the connection to the cluster to send commands and start a NASP job with the XML file. It then returns the ID of the job.

runNaspJobTest

The test calls runNaspJob and passes if runNaspJob does not throw an exception and returns an job ID.

Integration Testing

Integration testing ensures that the separate components of the GUI communicate with each other properly. Once we have checked the functionality of each separate class during unit testing we combine them together and run them as a group, luckily our project involves building a GUI so this already puts everything together for us. Integration testing is used to secure the integration between the view of the GUI, the inputs, and the intended output.

We chose to start with a more top-down approach, in that we started at the very top of the hierarchy and worked our way down to individual branches from there. Specifically we used a breadth-first process, in this case we started with the overall integration of the classes, fxml, and GUI as a whole. Once we were unable to visually see any obvious problems of integration between components we started to dig deeper into our interface and start coupling out components that made logical sense to be tested and working with each other.

Conveniently for the team, we are able to do some level of integration testing every single time we run the GUI from our local machines. Each time the run button is clicked the classes such as the MainController, the VisualizationController, the OutputParser, and so on are packaged all together with the fxml which dictates the look and layout of the GUI. In this way integration testing has been fundamental in our iteration process by testing and refactoring the way things are laid out within the user interface itself. In this way the whole team has been able to contribute to the integration testing portion of our project.

Some of the key things that were looked out for throughout the integration of the project were specifically how each piece of the GUI responds with one another. For instance, does the main layout fxml page display the components of the main window in a rationally sound way. We also need to know if a drop down list that is implemented throughout the fxml is grabbing the proper values, if any, from the main controller java class. Because we are fortunate to have explicit visual cues in our GUI we are able to visualize a problem between the integration of two or more components and go back and edit the source code accordingly.

The final steps of integration testing are currently underway and requires careful scrutinization in the way the user interface works with Tgen’s network. Because our GUI is currently sound in terms of stand alone integration, the real tests start to show in the form of the ability to local and network drives at Tgen, the ability to remotely access their clusters to create and send jobs, as well as grabbing and visualizing feedback, among other unforeseen problems. This process will not be extremely different than how the GUI was integrated amongst itself on a local environment.

Usability Testing

Usability testing is essential for the success of GUI based systems. In order for a user interface application to be successful, end users should be considered throughout the designing process. For or system, we focussed on the end-users experiences and on what they were comfortable with when we were designing our beta prototype.

We had a meeting with our sponsor and a number of his colleagues that were intended to use our interface where we interviewed them. We sat down with them and took a look on what tools they were using currently. We discussed with them what parts of the interface they liked and which they didn't like and why. We took notes with the intention of designing our prototype to have similar look and feel with the tools they already comfortable with. This process will help us design an intuitive interface that our audience would be very familiar with and hopefully would have a slight learning curve.

After carefully reviewing the tools that the Tgen employers are comfortable with, we began establishing patterns between them and began understanding why these interfaces work very well with our users. Figure 1 shows an example of one of the tools that tgen employers use. This tools gave us an idea of the basic structure they were comfortable with. We discovered that accessibility is key and our end-users like interfaces that has a lot of functionalities in one place. Their type of work requires them to use those tools frequently and for multiple tasks. Having multiple functionalities in one place enables them to save time and ease their work flow.

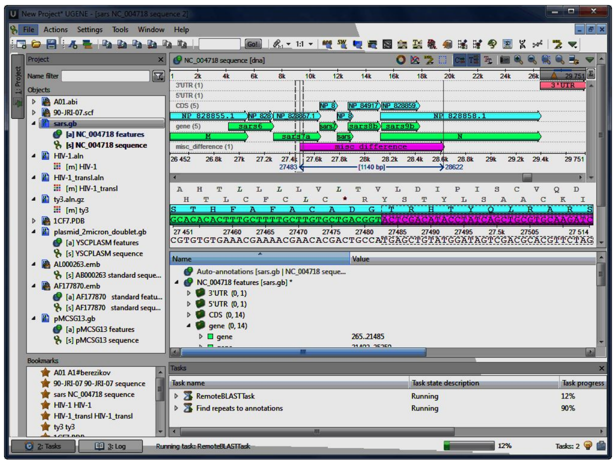

Figure 1. Ugene, a common informatics GUI

After we finished designing our beta prototype, we scheduled a meeting with our sponsor and a number of end-users. We presented our prototype to them and took notes of their initial impressions. We began modifying the design based on their feedback until they were satisfied. Including end-users into the design decisions made them familiar with the GUI and that would reflect very positively in the usability testing.

There are many techniques to test a usability of an interface. We are aware that the way we presented the prototype to the end-users may create biased opinions. Bias feedback is expected because the designers were in the same room with the end-users which may caused them to not say what they really think of the GUI and not give any negative feedback.

In order to solve this issue, we are thinking of having a two room testing environment where we design a lab manual and place the end-users in one room with the lab manual and record their interaction with the interface using video cameras. The designers will be in a separate room monitoring the testing process and taking notes without any interactions with the end-users. We believe that this process will reduce bias feedback significantly.

We will review the footage to try and discover where the users had problems with the interface and which aspects of the GUI they were having trouble with. This usability test will help us significantly to design an intuitive interface that Tgen and our sponsor would be very happy to use.