Software Testing Plan

Version: 1.1

Team: Orion
Sponsor: USGS
4/6/19

Isaac Shaffer

Brandon Kindrick
Chadd Frasier
Yuxuan Zhu

Table of Contents

Introduction

Our project is the Planetary Image Caption Writer and our task was to develop a tool that will assist researchers in using data from cub files to write captions and annotate images for later publication. Our tool allows researchers to upload an image made in the Integrated Software for Imagers and Spectrometers (ISIS) and extract the hidden metadata. This metadata can then be referenced in a caption that is either uploaded by the user or edited on the web page. The metadata can be used in the captions by placing a unique identifying tag in the caption editing section. The metadata can also be used to place helpful icons on the image. This tool will allow researchers to quickly generate publication ready figures.

No software project can exist without testing and our project is no exception. To ensure that our product works, securely and bug free, we have planned and implemented tests for our project. For testing purposes, we have decided to break up our project into three groups. The front end UI, the back end flask server and the Docker container. To ensure each of these modules is working properly, we have prepared a suite of tests for each. The front end tests will consist of testing user input and user interaction with the web app. These will focus on abnormal input and abnormal user interactions. Each front end element will be tested a minimum of three times for each type of abnormal input. The back end testing revolves around testing each of the functions and ensuring that no abnormal results are returned. This will be accomplished through unit tests of each function. There will be at least one unit test for each of the key functions on the backend. Finally, to test Docker, we have tests to ensure that ISIS is properly running, that the flask server is properly running and that users on any OS can communicate with all aspects of the product through Docker.

Our plan ensures that every aspect of our app is tested. Because our product is a web app, the two main components are the server that hosts the front end and the front end that the user interacts with. Most of our focus is placed on these components. The final module is containerizing the app so it is usable on any operating system. Because both of the main components will be tested at the point of containerization, the last tests revolve around making sure that Docker is properly hosting them and that they are accessible on linux, OSX and Windows. We will be testing every piece of the software described through unit testing in controlled environments, followed by integration testing that will ensure the server can communicate properly with the user, and lastly usability testing which will help us great a more useful application for our client and the other end users.

Unit Testing

Unit testing is the process of testing software by breaking the source code into separate modules and ensuring that they are fit for use in an isolated environment. The purpose of unit testing is to validate each piece of code to ensure it stands up to bad input and it never gives unexpected output before it is included in the whole system. When testing a class we must break the actions of the class into its own units and test them separately to ensure the whole class is stable. For the purpose of our project we will be breaking the metadata class into units as well as testing a few of the front end parser functions. Since the back end is written in Python we will be using the unittest class to test each function in the metadata class.

The first phase of unit testing will consist of testing the extraction algorithms that create the data dictionaries from the ISIS return file. Since there is no automated way to test if the file extraction is collecting every piece of data from the unknown pvl format we must do this manually. We will need to run the extraction algorithm on multiple cube files and compare the return file with the dictionary by hand. This will be a tedious testing set but we need to run it many times for the simple fact that it is not automated and will require much of our attention. In order to do this properly we will run this test on every test cube our client sends us in order to make sure that the cubes are properly extracted. We will run the extraction algorithm and print the resulting dictionary so we can test that retrieved data against the raw ISIS return file. The reason that this function is vital to test is because we will want every bit of data out of the cube file for later use and we must ensure we can get every line properly. This function starts the whole process off and it must be correct for our end product to be correct.

The next unit of the metadata section is the cleaning section which involves a dictionary clean, a key clean, and a string clean function that will remove and replace the unwanted or confusing characters in each function input. In order to test these specific issues we will write 3 different test values for each of the functions. Specifically for the clean dictionary function we will need to test many various dictionary entries including lists inside the dictionary and boolean values. The data we are retrieving is so complex that we will need to put the algorithm through as many test versions as possible. The dictionary does not need to be long in order to test so we will write short but complex dictionaries that will then be cleaned and assert checked against the clean version that we type out. For example, we will write a dictionary with many bad characters in the value sections of the dictionary and pass the dirty dictionary to the cleana dictionary function and assert it to equal the cleaned version of the dirty dictionary, meaning returning the dictionary with no unwanted characters in it’s values. We will perform the same operations for the string cleaning function and the key

cleaning function. The only function in this part that will need another form of testing is the key cleaning function. It is vital in our product that the key values always follow the priority order that is required to uniquely form key values for the dictionary. We will test this part of the unit by passing the key creation function different combinations of keys that all have a certain priority. This testing method will prove to us that our code is properly ordering the key values in a standardized way(IsisCube-Object-Group-Name-KeyName).

The front end unit will need to be check for inconsistencies in the dictionary parsing algorithm. Our front end takes the dictionary and breaks it into key and value pairs that can be used in the template editing section on our web app in the form of a custom key. In order to guarantee that the tags are unique and have the correct value we will run a series of tests on the algorithm. The tests will consist of passing the algorithm complex dictionaries that we formulate to test it's parsing capabilities. We will use more common structures seen in our product for one of the testing sets and some strange or incorrect data and we will handle them properly. The purpose of testing this function is to ensure that a dictionary of any form will result in a usable set of key value pairs. This is important for the usability of the application because if our application fails to create the key value pairs the user will not be able to use any of the data from ISIS.

The last big unit we will need to test the whole data class to ensure that the data from different instances of the web app do not become corrupted from multiple instances. In order to test this we will need to run our application on two different cub files at the same time and view the resulting data dictionaries in the order they are received. We will need to analyse the outputs of this test by hand because of the unpredictable nature of the data. We will be testing this with two, three, and four user instances to test the stability of the data class. This is one of the most important tests we have for the entire application because the whole purpose of using a web app, specifically a Flask server, is to allow for the power of ISIS to be used by as many people in as simple of a way as possible and if it cannot handle more than one data instance it will not be a very useful server.

Integration Testing

Testing each individual module is not enough to ensure that the project is working and bug free. Each module interacts with different modules, and these individual interactions need to be tested as well. If done properly, integration testing ensures that functions that interact between two modules have no security flaws, and return the proper data.

The first line of testing revolves around the front end module, and its interaction with the flask backend. The client's first interaction with the server is when it is instanced by the server. When the server runs, it starts an instance of the web client, and displays it to the user through a render template call. To ensure that everything is running properly. The first test being performed at this phase is ensuring that the client is initialized properly. We wrote a function that checks that each resource was properly sent to the page. Flask returns errors when a resource could not load. Our function simply ensures that Flask never throws errors when initializing the server.

The client and server send files back and forth, and it is crucial to ensure that every step of the way is guaranteed to work. The client's first page prompts the user to upload a .cub file and a .tpl file(optional). These files are passed to the Flask server and then stored. From this stored location, ISIS function calls are performed on the files and then the results stored on the server. Then when the user navigates to the next page, the results are then passed back to the user and displayed. To test this functionality, we had to test the interaction between both components. We started by implementing a function on the client that would fire when the client sent a file(.cub, .tpl, and both). We then tested this function with "normal" input and would verify that, when the function successfully sent a file to the server and returned "success", we would check the server and verify that the file was there and that it had uploaded successfully. We tried this with three different .cubs all with and without an optional .tpl file. After we verified that it worked with "normal" input, we moved on to testing abnormal input. We started with adding no files and attempting to upload. The client caught the lack of input and the testing function did not report a successful upload. We then attempted feeding it test .cubs that had minimal metadata and very strange resolutions. This did not affect the the testing function, and it the ISIS calls would return everything they could from the metadata.

On the next page, we needed to verify that the server was properly returning the results to the user. To verify this, we wrote a function that would compare the contents of the sent file, to the contents that the client received. With all of the input we gave it,

no matter how malformed, the client would receive the same data that the server had. This proved that the server and client were properly exchanging data.

Our app relays more than metadata to the user. The app takes in a .cub file and uses ISIS to return a png of the ISIS image. The Flask server then returns this to the client. To verify that ISIS was properly extracting the image and returning it to the user, we had to do a fair amount of manual testing. We did this by uploading a variety of .cubs, and then getting the output. We would then manually compare this image(.png) and the Image ISIS saved on the server. We could feed it any .cub and it would return the same png image. We tested it with three “normal” .cubs and three “abnormal” .cubs (.cubs with abnormal resolutions or metadata). Even .cubs that produced pngs with strange resolutions (500 x 2000) would work fine and return the proper png.

Finally, we needed to test the ability to export data from the client to the user’s pc. This is done very straightforwardly. The metadata, the .tpl and image are extracted from the .cub and are stored on the server. On the client, we added a function that pulls this chosen data from the server and saves it in the browser. To test that the exports worked, we manually compared the data extracted to the data stored on the server. Just like the previous tests, we tested this with three normal and three abnormal .cubs. We were able to get the proper files each time. The last two things we needed to verify could be exported were the finished caption, and the cropped image. Both were created by the client and not stored on the server in anyway. We were able to adapt the same export function to save the finished caption. We were also able to successfully save the cropped image to the client’s pc no matter what .cub we inputted. This proved all of the export functions work and client/ user pc interaction works.

Usability Testing

In this last section we will be discussing the Usability testing that we did for our product. What is “Usability Testing”? Usability testing refers to evaluating a product by testing it with representative users. Typically, during a test, participants will try to complete typical tasks while observers watch, listen and takes notes. The goal of doing usability testing is to identify any usability problems, collect qualitative and quantitative data and determine the participant's satisfaction with the product(Affairs, 2013).

In general, the process of usability testing involves a test moderator who gives test participants a series of tasks that they must perform with the design. The tasks represent actions that an end user would typically carry out with the finished product. During the test, the moderator observes each participant’s actions, often also recording the test session in notes. After analyzing the results of a usability test, the moderator reports on several points of interest that arose—these could include issues such as parts of the design that caused problems, the severity of these problems, as well as places in the design that the participants particularly liked(What is Usability Testing? , 2019).

For our product, we have created a web application for our client, who is a Research Geologist at the United States Geological Survey(USGS). So, to test our finished product, we need to find potential end users in USGS. Because our product is for publication use, we must discover end users who have space publication experience or future demand. Of course, we have a specific client - Dr. Laszlo Kestay, so he is the best end-user for our testing task at the beginning. And after that, we can try to find other scientists in USGS through Dr.Kestay, that will be much easier than us asking people in the building. Dr. Kestay will be able to point us in the proper direction of end users.

Our product is unlike any other product in this field of work. Before our product is created, Dr. Kestay wants to prepare images for publications by highlighting essential aspects of the image and including icons critical to the image, he has to load the same image into multiple processing software to achieve this. This progress is pretty tedious and our team created a web application that allows users to extract all metadata included in the cub file and manipulate the image at the same time, the user can now spend less time than before while still getting the same result and it’s thanks to our web application. However, because this is a new product, our users may not be able to familiarize themselves to the product immediately. So we will also be preparing the instruction document to help users along the way.

In addition, because our web application is used for the formal publication, the consequence of incorrect data is dangerous, if something is wrong in our web app, it will discredit the whole publication. So during the usability testing process, we must let users test the accuracy of our app by suggesting them to test .cub file that they have used in publication before, it will be an excellent way to examine our products abilities. Once more, as our client required, our web app will be able to run on the Windows operating system through a Docker container. The user must ensure they have installed Docker on their machines. If they do not have it, it will cause an environmental error. We have created a detailed instruction for Docker in our product and we will let end-users connect to a running server space or install it natively by reading the instructions.

After considering these prerequisites, we will design our specific testing plan in the following. We will test our product with four end users including our client; we will check with our client twice. One is at the beginning of our whole testing plan, another one will be at the end of our testing schedule. In the next two week, we will be testing with two end users. Our first user is our client, Dr. Kestay and we will gather three others. We will visit him at USGS next Monday and talk about our product. We will show him our current version and explain to him what he can do. Then we will let him use our product. Our team has three members, one member is responsible for asking him questions about our product, and the other two guys will be responsible for recording everything. During the testing, our team will be recording everything that Dr.Kestay feels was either, surprising, not useful, or excellent. And we will ask him his feeling when he tries each function of our product. We will let him start using our product from the very first step, installing the Docker. After interviewing with him, we will do the analysis, for the things he feels disappointed about and we will take the criticism positively and improve on our product.

After that, we will interview another scientist at USGS on the same week and we will show him the revised version. We will use the same plan that we used with Dr. Kestay. After testing with this new scientist, we will do modification according to his/her feedback. For a new testing week, we will examine another new scientist, and make modifications like the week before. Finally, on April 19th, we will revisit our client to show him our final version. We will let our client try our product from the first step as before, we will observe his behavior in the specific times that he showed displeasure before in the previous testing cases. We will finish this off by interviewing him about his feelings on the changes and make any last minute fixes based on his feedback.

That’s the whole plan for our usability testing. Using the process described we will be able to get each end users’ feedback through the testing and make improvements according to their feedback which will be vital to the usability of our product to the people that will use it the most. After this testing, we believe our web

application will become more friendly to users and let them feel happy and efficient when using our product, that is the purpose of implementing this testing.

Conclusion

As we know, a good test plan outlines activities that are aimed at ensuring that our project's implementation exhibits the necessary functional and non-functional characteristics without error. This test plan documentation is to ensure all Functional and Design Requirements are implemented as efficiently as possible and we will do it using the tests in this article. For our project, we will utilize all three testing types; unit testing, integration testing, and usability testing.

Unit testing is a level of software testing where individual components of the software are tested in an isolated environment. The purpose of unit testing is to validate each unit of our product separately and ensure they perform as designed. A unit is the smallest testable part of any software(Unit Testing, 2018). The product of our team is a web application. Since it is a web app, it should have two ends: the frontend and the backend. We must ensure our unit testing plan can cover both sides. To test the front-end of our web application, we will break the metadata class into units as well as checking a few of the front end parser functions. The important steps included in the testing include; testing the extraction algorithms that create the data dictionaries from the ISIS return file, testing the cleaning section. Then we will be testing for inconsistencies in the dictionary parsing algorithm by checking that keys and values are the same as those in the extracted pvl file. On the other hand, we also need to test the backend of our web application, and it will be a little easier than checking the frontend, because our backend is written in Python, we only need to use the unittest class to examine each function in the Python data class.

After that, we will try to implement integration testing for our product. Integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units (Integration Testing, 2018). During this testing phase, we want to combine the individual tests performed in unit testing to see how they work together. In the unit testing, we have checked with functions of frontend and backend separately, now we want to integrate them and test it as a whole product. We will test our entire web application from uploading cub file. We will check if the file uploaded successfully, if the data extracted correctly, and then check if keys and values can corresponding to the data retrieved. Finally, we will test adding annotations to the image our web app extract, to make sure we can manipulate the image and then export the image, the csv data, and the caption text file. The whole procedure needs participant of both frontend and backend. If this test works well, it proves we have a stable product!

Finally, we also need to implement a technique used in user-centered interaction design called "usability testing" to evaluate a product by testing it on potential end

users. This can be seen as an irreplaceable development practice since it gives direct input on how real users interact with our product. To implement usability testing, we will interview four scientists who are working at USGS now, because our product is designed for scientists who work on space publications. We will invite these four scientists to answer questions about the design, and then test our real product from installing the docker to set up the environment. We will also give them instructions we write for end users to help get started with our web app. We will invite our client twice during this testing phase, because he is the specific person who will use our product daily, his suggestions are the most important. After inviting him to use our product at the beginning, we will improve our web app according to his feedback. Next, we will test it with two other scientists, and improve the practicality and ease of use gradually. Finally, we will invite our client, Dr. Laszlo Kestay, to use our product again and get feedback from him. After talking with him Dr.Kestay one last time, we will make the last change to our code for this phase of development. Lastly, we will deliver our whole product to our client and possible work with him in the future.

References

Affairs, A. S. (2013, November 13). Usability Testing. Retrieved from https://www.usability.gov/how-to-and-tools/methods/usability-testing.html

What is Usability Testing? (n.d.). Retrieved April/May, 2019, from https://www.interaction-design.org/literature/topics/usability-testing

Unit Testing. (2018, March 03). Retrieved from http://softwaretestingfundamentals.com/unit-testing/

Integration Testing. (2018, March 03). Retrieved from http://softwaretestingfundamentals.com/integration-testing/

Usability testing. (2019, February 21). Retrieved from https://en.wikipedia.org/wiki/Usability_testing