Software Testing Document

Team Clean Carbon

April 7, 2023

Sponsor: Allie (Alexander) Shenkin

Mentor: Vahid Nikoonejad Fard

Members: Curtis McHone, Richard McCue, Shayne Sellner, Justin Stouffer, Jonathan Bloom

Version 1.0

Accepted as baseline testing steps for the project:

Sponsor Signature

Team Lead Signature

Table of Contents

Introduction

Climate change has become a growing concern as the overabundance of carbon dioxide and other greenhouse gases in the atmosphere causes a rise in global temperatures. To offset their carbon footprint, big companies invest in climate change projects by purchasing carbon credits. The carbon market is relatively new and not strictly regulated; it was valued at $500 million in 2020 and is predicted to grow exponentially to $90–480 billion in 2050. However, the market is not easily accessible to consumers and companies, and estimates about reforestation statistics must be obtained by contacting individuals who manage reforestation plots. Our sponsor Allie Shenkin's research finding helps consumers estimate reforestation statistics and make carbon credits 30% more profitable, motivating developers to invest in reforestation projects. Allie and his team developed a proof-of-concept tool to implement this new research finding. However, further development is necessary to create a more reliable and efficient solution. Our team was tasked with this issue so that we may create a more robust, reliable, and faster-working tool to bring a new level of transparency and accessibility to the carbon market.

To ensure the accuracy and reliability of our solution to this issue, we need to set up a software testing suite. Software testing is a critical aspect of software development that assesses the quality and functionality of software applications. This will allow the developers to identify any defects, errors, or gaps in the software's functionality. Software testing can be done manually or through automated tools. This document will outline our overall plans for software testing within our project.

To ensure the quality of our project, we will employ many different software tests, ranging from login access to data consistency across the entire project. We will first start by ensuring that our login information is secure and cannot be accessed by nefarious methods, as well as ensuring that someone who is not registered with our service cannot gain access to it.

On the front end of the project, we need to verify that the polygon coordinates and area are correct, as well as ensure that the year is not different from the one the user chose. In the backend of our project, we will test to ensure that the polygon that we are cutting out of the raster is the same as the sliver of the raster that we receive. We will also make sure that the average value of this raster sliver is accurate by conducting manual testing. Because of the nature of our program, most of our testing will have to be done manually, as the coordinates calculation of the polygon, the passing of data between the front and back ends, the cutout of the raster, the average value calculation, and the total carbon uptake calculation will be automated and do not require separate input from the user. Due to this, we will need to ensure each part is working properly and does not create an issue that moves down the line of computation. We will then test and ensure that the data we are communicating between the front and back ends of our project is consistent and accurate. Due to the connection between the front and backend and how much the front and backend rely on each other, each will be heavily tested to create a robust product.

Unit Testing

Unit testing is a process done in software development that allows for the testing of specific pieces, functions, or function calls. A unit is the smallest testable piece of software. Most unit tests consist of three main stages: planning, analyzing the cases, and writing the tests. Testing a unit allows the software to be reliable and robust. Unit tests work by using predetermined values to compare against the values of the output of the software unit. The software unit will operate on these predetermined values and return the output. The unit test will then compare the output to the value the developer gave the unit test. If the values match, then the unit passed; if the values don't match, then there is a bug or error within the unit. To ensure there are no errors carried forward within the code, each unit is tested individually, as well as provided with different inputs depending on what the unit needs to be tested for.

Our goals for unit testing our software are to ensure that our code is accurate and reliable. Some areas of our software that we know may not be reliable and need to be tested are the correct coordinates, map projection, CO2 uptake amount, and the year of prediction selection. Our hopes for writing and implementing unit tests are to find bugs within our code, confirm our polygon coordinates, and ensure our predictions are accurate according to our rasters. Once every unit of the software is tested and proven accurate, we can say that our software, at its core, is reliable. Integrating these units together is still another area of concern, which will be discussed later.

One of the software tools that we are going to use to support our unit testing is included in our Django framework, and it uses the standard Python library, unit test, to create the unit tests. Django implements its own version of Python's test library, which makes it easy for us to use, as we do not need to add any more dependencies to our code.

There are a few bigger units of our software system that we want to focus on for unit testing, but more generally, we want to test every individual unit of the software, no matter the size. The frontend of our website has a few units that need to be tested. These are ensuring that the polygon coordinates are accurate compared to our projection and that the area measurement of the polygon is accurate. Our backend also has some units that need to be tested. The main unit that needs to be tested in the backend is ensuring that our prediction system is computing the correct number of carbon credits. This is the largest part of our backend, and it is going to need the most testing by far.

As the frontend is in javascript, HTML, CSS, and OpenLayers, it is much harder to set up robust unit tests compared to the backend, which is just Python. To test for correct coordinates, we will have a specified polygon that we know the coordinates of, and then draw the same polygon on our software and use a unit test to check that the coordinates are the same. Along the same lines as the polygon coordinates, we need to make sure the polygon tool and map view are using the same projection of coordinates so that the user doesn't draw a polygon where they don't intend to. In order to test that area calculations are correct, we will have a predetermined polygon whose area we know and then use our area tool to calculate the area. We will then use a unit test to check if the areas are the same.

Unlike the frontend, the backend is written purely in Python, using the framework Django, which makes unit testing a bit easier in most cases. Just like the frontend, we cannot take advantage of everything that unit testing has to offer, as we need to do more static analysis outside of our software. The main component that needs to be tested on the backend is our global prediction system. We need to ensure that the results that the global prediction system is coming up with are as accurate as possible. The way we are going to test this is very similar to the frontend. We are going to get a polygon from the frontend and bring that into QGis. We will cut out the area in the global raster that the polygon covers and do the global prediction on that plot of land by hand. If the results that we get by hand are the same as the results of our global prediction system, we can assume that our global prediction system is accurate.

The final step in unit testing for our project is to compare all of our results against the prototype software that our project sponsor has created. He has mentioned that this prototype is not perfect, however, so this comparison will be another layer of testing, but it won't be nearly as accurate as the methods stated above.

Through the use of all of the testing methods mentioned above, as well as further testing regarding invalid inputs, we can prove the robustness of our software units. Once each unit of the software is tested to exhaustion, we can be confident that each unit is correct and ready for integration testing.

Integration Testing

Integration testing is the procedure of making sure that the interactions and data exchanges between the major modules of our system are occurring correctly and maintaining accuracy. During our integration testing, we need to make sure that all of our functions are returning the correct value types (string, boolean, integer, character, etc.), as well as ensuring that all of the features of our application are connected to one another. A goal of our integration testing would be to make sure that the user can access all aspects of our program seamlessly without causing any unexpected issues. Another goal of our integration tests is to ensure that each component of our software can effectively communicate and work with each other without crashing or creating errors. In order to achieve this, our integration testing will start by analyzing the inputs and outputs of all our Javascript and Python functions. This way, we can rest assured knowing that the transmission of our data is not an area of concern.

As a team, we have come up with many different integration tests. The first test that we will conduct revolves around our PostgreSQL database and the Django framework. Our team needs to ensure that when somebody attempts to login, query results are being sent over to Django. In order to test this, we will display all the responses from the PostgreSQL database in our Django framework. This test is important because we don't want registered accounts to get an error when trying to login, preventing them from using the rest of our pipeline. We also need to make sure that only registered accounts in our database get access to the main page. In order to test this, our team will try to login with various usernames and passwords to see if we can bypass the code logic. This test is extremely important because once we transfer our website to the public domain, anybody with the URL link would theoretically be able to access all portions of our website regardless of whether or not they have an account.

The second integration test that we will conduct is on the logout component. The logout component will communicate with the main page and the welcome page. This component needs to be able to restrict the users' access to the main page and bring them back to the welcome screen, further protecting the program pipeline. In order for our team to conduct this test, we will attempt to login and logout of various valid accounts. After conducting that, we will then check to see if we can access other portions of our website through buttons and direct URL links. In order to make sure that this integration test is being conducted accurately, we need to print the logged-in user's information at various points in our program. This way, we are sure that the user has the correct credentials to access our website as well as ensuring that they cannot access any aspects of our website illegally.

Our third integration test that we will conduct works on the prediction page of our program. This integration test will ensure users that they cannot access any aspects of our pipeline directly. This test is important because the user would be able to crash one of our website pages if the prediction is run without any data selected. In order to test this, our team will try to input various keyboard commands and mouse clicks throughout the main page to see if there are any vulnerabilities. We will also try to hardcode the prediction page URL into our browser to see if we can get illegal access to the prediction page.

Our fourth and final integration test will check to see if the frontend data from the main page is being accurately sent over to our backend scripts. The goal of this test is to make sure that all coordinates and years that have been selected are transferred when the user attempts to run the prediction. In order to conduct this integration test, we will add print statements that display the year selected as well as the polygon’s coordinates in both our frontend Javascript and our backend Python. This integration test is important because we want to ensure users that our frontend code can accurately and efficiently communicate with the backend code.

After conducting these four integration tests, our program's components should be connected and communicating as intended. After we successfully test and debug all the aspects of our integration tests, our program will be one step closer to a finished product.

Usability Testing

Usability testing is testing that focuses specifically on the interactions between the software system and the end user. It is meant to ensure that the users are able to effectively use the software and access its functionality. This works by having users test our system and provide feedback on certain features or areas that need improvement and other features that they feel are executed well.

To begin the usability testing, we will first consider what types of users our software will be providing functionality to. Our software is first and foremost a tool to determine the estimated amount of carbon credits someone can receive if they choose to reforest a specific plot of land, and the system is also secured so that only authenticated users will have access to the software. Therefore, the users that we can expect to be using our software must be somewhat experienced in the carbon credit market, reforestation industry, and/or carbon emission research. These types of users will have a clear understanding of the functionality that our software provides, and in turn, they will be looking for specific data results when using our software. Given this user background, we will be able to conduct our usability testing accordingly to receive the best feedback.

The first set of users that we will have test our software will be chosen from a group of users that have a strong background in reforestation, carbon emissions, carbon uptake, carbon credits, and similar backgrounds. These individuals will already have an idea in mind of what our software should look like based on their pre-existing knowledge in these areas. Before they are able to use the software, we will have them explain to us how they would like the software to be implemented, so as to avoid any bias once we start testing. This will help us understand what the experienced user will expect when they are going to use our software. This user group will be extremely beneficial to our usability testing, as they will be the main set of users who will use our product.

The second set of users that we will have test our software will be chosen from an average group of users, who do not have a background in any of the areas mentioned above in the first group. They will have little to no experience dealing with carbon credits or reforestation, and will be using our software for the first time with no expectations or previous knowledge. We will give them an overall introduction to our product and what it does, but we will not explain how to use any of the features. This testing will help us understand what the average user expects from our software, and they will be able to provide certain insight that the first group of users might overlook. This could be very simple things such as button placement, popups, and overall ease of use. This user group will also be very beneficial to our usability testing because they will provide better insight into exactly <u>how</u> the software functions, without being biased about what data the software is returning.

For each of the group tests we will be conducting, we will conduct and measure the testing as follows:

1. The team will meet at the SICCS building with the group of users.
2. A room will be reserved for the testing so that the user is not distracted.
3. The user will be given some information about the software product.
    a. For group 1, we will provide information about what software result is being returned to the user, how this data is extracted, and how the toolbar works.

    b. For group 2, we will provide only basic information to the user, that being what the overall point of the software is, and some basic information on the toolbar.

4. As the user is testing the product, we will ask them to use the “think aloud method” where they will talk aloud about everything they are doing and thinking as they are using the product. This allows us to have a great deal of insight as to the user thought process when they are using the product.
5. The team will be writing down notes and recommendations about what the user says as they are testing the product.
6. Once the user feels they have a good understanding of the software, we will ask them directly how they feel about certain features and aspects of the product. The team members will be taking notes and asking follow up questions if needed.
7. Once the testing is complete, we will thank the users for their time, and finish the usability testing.

The usability testing plan for our product involves a wide range of users that will allow our group to receive feedback from multiple perspectives and points of view. This will be very beneficial to the development of our software, as we will be able to adjust the software as needed to improve the overall ease of use and accessibility. This will happen in the next couple of weeks, so we will have enough time to implement new features or adjust existing features as needed.

Conclusion

With our sponsor's research findings and the proof-of-concept tool that they developed, our team has been tasked with developing a robust and user-friendly tool that brings much-needed transparency and accessibility to the carbon market.

In order to ensure the accuracy, reliability, and security of our solution, we recognize the need for comprehensive software testing. Software testing is a crucial aspect of software development that evaluates the quality, functionality, and effectiveness of software applications. Our testing suite will encompass a range of tests, from login access to data consistency across the entire project.

Our primary focus is on ensuring the security and privacy of user login information, preventing unauthorized access to our service, and computing accurate reforestation statistics so as not to mislead our users.

On the front end of the project, we will conduct extensive testing to verify the accuracy of polygon coordinates, areas, and year selection. In the back end, we will test to ensure that the polygon cut out of the raster matches the received sliver of the raster and that the average value of the raster sliver is accurate. Due to the automated nature of many aspects of our program, much of our testing will have to be done manually to ensure proper coordination between different components of the software. In summary, our goal is to develop a reliable, efficient, and secure tool that meets the needs of users and facilitates greater accessibility to the carbon market.