Software Testing Plan

3 April 2020
Version 1

Sponsors

Dr. Kiona Ogle

Dr. Michael Fell

Mentor

Isaac Shaffer

TreeViz

Riley McWilliams
Qi Han
Haitian Tang
Daniel Rustrum
Alex Bentley

Contents

1 - Introduction

TreeViz is developing a user-friendly platform for visualizing tree growth. Our clients, Dr. Kiona Ogle and Dr. Michael Fell, came to us with the ACGCA model: a program used to realistically simulate the growth of trees over time. While the model is currently used to study tree growth in an ecological research context, our clients wish to expand this audience to the general public so that anyone can learn about tree growth. However, the program is command-line based, only outputs raw numerical data, and requires knowledge in tree biology to understand the inputs and outputs. It is also not available online, which limits who can use it.

Our goal is to deliver a web application that wraps the ACGCA model with a more user-friendly method of submitting input and receiving output. It will be hosted online, so that anyone with an internet connection can use it. The 30+ input parameters will be entered by the user via text boxes rather than a command line. They will also be organized on a web page and include descriptions, allowing anyone to better understand what they do without a background in biology. The output will be displayed as a graphical representation of the simulated tree. While the raw numerical data will also be available, this graphical tree will provide a more interesting way for the user to learn how certain inputs affect a tree's growth. Implementing all of these features will make the ACGCA model significantly more accessible by the general public, allowing more people to utilize it and learn about tree growth.

With a completed alpha prototype, this document serves as a plan to follow when testing the product against the specified requirements (see Requirements Specification document). This is important for ensuring that the final product that we will deliver not only meets the requirements of our clients, but also does not function incorrectly when exposed to unintended environments. For example, if the user enters an input value that is much higher than the intended maximum for a given parameter, the software must be able to handle it and give an error instead of crashing.

Our testing plan consists of three major components: unit testing, integration testing, and usability testing. First, unit testing will be conducted on the individual parts of the project, including the user login and survey pages, user input and visualization page, and model manager. To do this, we will test each function to ensure that it works as intended, providing the correct output given intended inputs. Second, integration testing will be used to ensure that the components work together properly as a whole system. We will start by running the whole system as it is intended to be, to make sure it works correctly. Then, we will check for errors by running the system in an unintended way, which will show us where possible pitfalls in the software may lie. Third, usability testing will provide us with insight on how users will be utilizing the system. We will ask people to try our system and take note of how they are using it. Our testing process will largely be focused on usability testing due to the user-driven nature of the product. As such, we will put more weight on the results of the usability tests than the unit and integration tests.

The first part of our testing plan is unit testing.

2 - Unit Testing

Unit testing is a method of ensuring that the product's individual functions work the way they are intended to. It involves conducting tests on each function by providing various inputs and checking if the output is correct. Functions that pass the test will allow us to prove to our clients that they are working as envisioned. Functions that fail the test will show us, as developers, where improvements need to be made.

Our goal with unit testing is to confirm whether or not each function works as intended. Doing this will make us aware of any bugs in the software, so we can fix them as soon as possible. With each correctly working function, we move closer to the final product that will be delivered to our clients.

Unit testing will be conducted in parts to simplify the process. The first part is made up of the user login and survey components, which are written in JavaScript. We will be using a JavaScript testing framework called Mocha to perform the necessary tests. The second part, the visualization component, is also written in JavaScript, so we will use Mocha to test it as well. The third part, the model manager, is written in Python. To test this, we will use Python's built-in Unittest module .

2.1 - User Login and Survey

The user login and survey pages are where the user will create a new account, log in to an existing account, and fill out a survey for analytics. They are all written in JavaScript, so we will use Mocha as a testing framework for their methods. Part of the user login and survey functionality interacts with our Firebase database by transferring data between it and the webpage. As such, some expected outputs may be related to Firebase. The functions are listed below with a brief description of what they do to provide context, as well as their intended input and expected output.

login()

When users login to the webpage, this function will be called in order to allow the user to sign in with their email and password. When the user clicks the login button on the page, this function will be called.

Intended inputs:

The user's email
The user's password

Expected outputs:

This function will send the email and password to the Firebase database. If the password in the database matches the corresponding email, then the webpage will be redirected to the input page. If an error happens, such as an incorrect password, an error message will be shown.

submitName()

On the survey page, when users finish the survey and click the submit button, this function will be called.

Intended inputs:

The survey fields:

First name
Last name
Locale
Affiliation
Affiliation role

Expected outputs:

The survey data will be sent to the Firebase database under the user's name.

LogOut()

When the user clicks the logout button, this function will be called.

Intended inputs:

None

Expected outputs:

The current user will be signed out and the page will be redirected to the sign-in page.

SignUp()

When the user fills in the email and password on the sign-up page and clicks the submit button, this function will be called.

Intended inputs:

Correctly formatted email.
A password that is longer than 6 characters

Expected outputs:

This method will create an account on the firebase server with the email and password. Then the page will be redirected to the survey page.

2.2 - User Input

The input part of the input-visualization page is written in JavaScript. We will do the unit tests for this part using the Mocha testing framework. There are also some parts that interact with our Amazon Web Services (AWS) server. We will conduct the unit tests based on the reaction that we get from the AWS server in these cases. The functions are listed below with a brief description of what they do to provide context, as well as their intended input and expected output.

postData（）

This function will be called when users hit the post-data button.

Intended inputs:

All of the 36 inputs necessary to run the ACGCA model.

Expected outputs:

The website will send what users have entered to the AWS server.

checkValidity()

This function will be called when isDisabled() is called in order to check the validity of whatever users have entered in the input fields.

Intended inputs:

All of the 36 values in the input fields.

Expected outputs:

It will return a boolean value which represents whether or not users entered valid input. If the inputs are all valid, it will return false. Otherwise, it will return true.

isDisabled()

This function will be called when users enter or change data in the input field.

Intended inputs:

A boolean value which is returned by checkValidity().

Expected outputs:

When the input is not valid, the “post data” button will be disabled. If the input is valid, the “post data” button will be enabled.

2.3 - Visualization

The visualization component of the project relies heavily on functions built in JavaScript and a 3-dimensional graphics framework called Three.js. Three.js creates a renderer that is attached to an html div, allowing objects to be displayed in it for the user to see on their web page.

Using Mocha, we will assert tests on each of the functions that make up the visualization component. These functions are listed below with a brief description of what they do to provide context, as well as their intended input and expected output.

getData()

The getData function is called when the user requests the output data. It calls the AWS server and requests the output data of the model.

Intended inputs:

None

Expected outputs:

A json object that contains the data that is returned from the ACGCA model.

initialize()

The initialize function is called when the page loads. It creates a Three.js renderer and appends it to a div in the html. It also sets up 2 scenes, one to display the trees, and one to display the tree rings.

Intended inputs:

None

Expected outputs:

An empty, rendered scene that is displayed on the page.

drawTree()

The drawTree function is called when the html timestep slider is moved. It erases the scene of all objects (the previous tree) and draws the tree at the current timestep. The tree is made up of a trunk object created from a cylinder, and a crown object whose shape is changed based on the crownShape variable.

Intended inputs:

The outputs of the ACGCA model.

Expected outputs:

A tree made up of a trunk and a crown object.

drawRings()

The drawRings function is called when the user presses the “RINGS” button. It draws a cross section of the tree’s trunk at each timestep.

Intended inputs:

The outputs of the ACGCA model.

Expected outputs:

A series of concentric circles with radii based on all the time steps of the tree.

setCrownShape()

The setCrownShape function is called when the user presses one of the menu buttons to change the shape of the tree’s crown. It takes in the selected crown shape as a parameter, changes the variable for the crown shape, and then calls drawTree() to redraw the tree with the correct crown shape.

Intended inputs:

“cone” “sphere” or “cylinder”

Expected outputs:

A new tree with the correct crown shape.

setScene()

The setScene function is called when the user presses one of the menu buttons to change the current scene. It takes in the selected scene as a parameter, displays it on the page, and hides the other scenes.

Intended inputs:

“treeScene” “ringScene” or “rawDataScene”

Expected outputs:

The correct scene is displayed on the page with the other scenes hidden.

2.4 - Model Manager

The Model Manager component of the project is a composite of three individual systems working with each other, the inputQueue and outputCollection within AWS, and the ACGCA wrapper which is run locally. However these systems are all written within python and can be tested with python.

Using the Unittest module within python, we will assert tests on each of the functions that make up the Model Manager component. These functions are listed below with a brief description of what they do to provide context, as well as their intended input and expected output.

inputQueue()

The inputQueue functions are called when the inputs from the website are sent to the API and when the model needs the values. On one end of the queue, it takes in any json formatted object as the request body, then it returns a run ID for the user. On the other end of the queue it does not take any inputs and returns an escaped version of the inputs submitted from the first end of the queue.

Intended inputs:

The user-submitted data from the input page on the website.

Expected outputs:

Run ID

Escaped version of input

outputCollection()

The outputCollection function is called when the ACGCA model is done processing and when the user wants the results. On one end of the queue it takes in any json formatted object as the request body. On the other end of the queue it takes the run ID provided by the inputQueue and returns an escaped version of the inputs submitted from the first end of the queue. It will return an empty object if nothing is in the collection with the run ID.

Intended inputs:

Output from the ACGCA model

Expected outputs:

Escaped version of the output

runWrapper()

The runWrapper function is called when a run command is called. Once called it will continue to run until the process associated with it is stopped. This is tested by ensuring that the program will take any proper inputs from the inputQueue and put the proper results in the outputCollection.

Intended inputs:

A JSON-formatted structure of the inputs that the ACGCA model is expecting

Expected outputs:

A JSON-formatted structure of the outputs from the ACGCA model

runModel()

The runModel function is called within the wrapper. Similarly to the runWrapper function, it expects a proper input and returns the correlating result. However, the function will be tested directly, so there are not any problems in dealing with processes.

Intended inputs:

A JSON-formatted structure of the inputs that the ACGCA model is expecting

Expected outputs:

A JSON-formatted structure of the outputs from the ACGCA model

3 - Integration Testing

Integration testing is a level of software testing where individual units are combined and tested as a group. It is based on unit testing and combines units as a whole system. The ACGCA model functions as a black box; it receives input and returns the output. To apply integration testing, we will mainly focus on data passing. If the data is successfully passed from the website to the model to be calculated, and then it can be passed from model to tree visualization, our product succeeds in passing the integration testing.

The purpose of this testing is to expose faults in the interaction between integrated units. We are going to apply integration testing on the whole product first to see if it functions as intended. Then the testing needs to be divided into several more detailed parts. The basic approach is to offer the input and do the black box testing. Then we can check whether the output is what we need. As our clients told us, random data may violate the limitation for variables. In this case, to avoid troubles caused by using random data, we asked our client to provide default data. We will use the default data as our input for the rest of our testing.

To make the testing more organized, we divided it into four parts: user input to website, website to model, model to tree visualization, and the system as a whole.

3.1 - User to the Website

This part focuses on the integration between user input and the data that we send to the model. It focuses on whether users enter their data correctly. We have already set limitations to the user input boxes to account for erroneous data. If the user enters some values which are out of range, an error message and help box will show up to clarify the limitation.

To test whether we can pass data from the user to the model, the data which is going to be passed to the model will be stored. Then we will compare whether the data we store is the same as the default data. If there is no difference between them, there is nothing wrong with the data passing from user to the website.

3.2 - Website to Model

This part focuses on data passing from the website to the model. We are using AWS as a platform to help with the data transferring. Because our client gave us the model and there is no need to change it, we will treat the model as a black box. We have already created a wrapper for the model.

To test whether data passing is successful or not, we will make a function in the wrapper to print the data that the wrapper receives. If the received data is the same as default data, then the data passing from the website to the model is successful.

3.3 - Model to Tree Visualization

This part will test whether we can get the data transferred from the model to the tree visualization. Because the model is a black box, we will directly get the output from the model once the wrapper receives the data. The wrapper will pass the data it receives to the model, which will run and return the output. The wrapper will receive the output and pass it to tree visualization.

To test the data passing, we need to test the pass between the model and the wrapper, and the pass between the wrapper the and tree visualization. First, to test the pass between the model and the wrapper, we need to compare the output data of the model and the received data of the wrapper. If they are the same, data passing from the model to the wrapper is successful.

Second, we need to test whether the passing between the model and the tree visualization is successful. Because the output data consists of arrays, we need to ensure that the passed data includes multiple elements. In the tree visualization, there is a moveable range slider which represents the time in the tree's growth. In this case, if we move the slider, the visualization should change along with it. If we can see the difference of visualized trees via moving the slider, the data passing of arrays is successful.

3.4 - The Whole System

This part focuses on testing the whole system. We will have some other input data and run the whole system with this new data instead of the default values. We need to make sure that the new data we are going to use will not violate the limitations of the input data.

After we enter the new input data, we will wait for AWS to retrieve it. Then, AWS will send it through the ACGCA model and get the outputs. Finally, we will check tree visualization. The render of the visualized tree should show up and we will move the slider to test whether the visualization will change with it.

4 - Usability Testing

The integration between the many aspects of our website is something that is very important to the usability of it. If something is not correct there, then the user will definitely notice. We want to provide the user with the best experience possible, with little latency and maximum appeal. As developers for this project, we are too familiar with how it works, and have become blind to the flaws that a typical user will see. We aim for the average person or data scientist to be able to navigate the website with ease, enjoy each page's layout and style, and understand each input that they put in and how that affects the output that they receive.

Usability testing is the process of finding out how usable the product is for our perceived audience. Through the tests and data gathered, we will be able to form a general consensus of what people think of our website and what we should improve to promote the use of the product. Before we decided on how we were going to create these tests and gather the results, we first needed to understand the product. We are creating a product that is intended for individual use. It will be hosted on a website, and it is aimed at the average person, student, teacher, or biologist. Since this will be an individual tool it will only need to run a single instance of the model at a time. Therefore, we will be able to test this product on one person at a time rather than a group. This will allow us to find more honest answers when asking individuals what they think of the product, contrary to a group setting where people may change their answers depending on their peers. This product will be for the average person, so we will have to create something for that type of individual and whether or not this would be appealing to a person who is not a biologist; that being said we will want to gather feedback from biologists, as well, in order to make this a serious tool for people to use. Knowing what we are making and for who is the best information that we can have in order to test appropriately.

Test Flow

The usability testing will be split into two individual user tests. It will be individual because that is how the product is designed to work, even in a group setting. The overall goal of the first test will be for the user to be able to navigate from the beginning to the end of the product; the beginning being their desktop and the end being the visualization. All notes that we take will be recorded by a testing proctor. We will have each user follow the steps below:

First Test

1. The user will be given brief instructions on what they will need to do in order to complete this test and what they should expect on completion. They will be given a url and that is it.

Upon completion of the first test the user will be asked what they thought of it: what they liked and did not like, where they got stuck and how, what they think could be better and how, etc. The information that is gathered during this stage will be recorded, and the user will begin the second test.

Second Test

1. We will first have the user navigate to the landing page. We will ask the user what they thought about this, whether or not they want to be able to find it on Google, whether or not they thought the url was easy enough, etc.

2. Once the user is on the landing page we will ask them to navigate to the 'returning users' section of the webpage. Once the user has done this we will ask them what they think about the look of the page. How easy was it to get to the page? Did they recognize this during the first test? Is the login function easy to understand? Could they see people coming back to this website or is this functionality irrelevant?

3. We will have them login and be redirected to the visualization page. What do they think about the visualization page? Are there too many inputs? Are they hard to understand? Do they like the visualization? If not, what do they not like about it? Are the raw data outputs easy to understand? If they could change anything about the page what would it be?

Once they have finished the second test, we will again ask them what they liked and did not like. Was there anything they noticed that they did not before? Was it more clear this time when walking through the website, the same, or less? Would they recommend this to someone even if that person had no interest in biology?

Regime

Once we tested several users we will need to assess the data that we have gathered. We will start by taking the notes from each person and comparing each test's notes. Once we have consolidated the user's feedback from both tests, we will go through each part of the website and try to find correlations between different user's feedback. From there we can find the most prominent problem for each aspect of the product. It will be at our discretion to account for outlying comments and criticism from users if they are not shared with the group. We will be mainly concerned with shared criticism to better account for the general opinion of our product moving forward. The data that we end up with will be use cases that we can utilize to mitigate risks and improve our product and ensure a usable and appealing website. We foresee completing the usability testing process by April 14th.

5 - Conclusion

TreeViz is nearing the end of development of a user-friendly platform for visualizing tree growth. With all of the main components completed, we will need to test them and assure our clients that the product they envision will be delivered. With data on the state of the system as it currently stands, we can move forward with more focus on what needs to be fixed and what needs improvement.

Through conducting unit tests, we will be able to focus on functions that are not working as intended and fix them. Integration testing will help us find problems in the functionality of the system as a whole, and catch errors that may occur when transferring data between major components. Usability testing will allow us to gain a better understanding of a user's perspective of the product. With this information, we can focus on improving features that might not work well in an actual use case and keep what already works well.

With an emphasis on usability testing, we will be able to improve the product from the user's point of view. This is one of the more important aspects of the product, due to its user-driven nature. With these improvements, we are confident that our clients will be pleased with the product they recieve.