Software Testing Plan

04/01/2022

Version 2.0

Team Biosphere

Project Sponsors: Dr. Christopher Doughty, Jenna M. Keany,

Team Mentor: Melissa D. Rose

Team Members: Matthew Nemmer, Brandon Warman, Teng Ao, D'Yanni Bigham

Table of Contents

Introduction

Tropical forests play a vital role in large-scale environmental processes and are some of the most biodiverse ecosystems. They help stabilize the Earth's climate and "contain over 30 million species of plants and animals" [1]. Therefore, tropical forests need to be preserved and protected, a job that is partly the responsibility of forest ecologists. They determine the environmental importance of the forest, which allows policymakers and conservationists to better understand environmental changes and enact legislation to protect forests. To do this, forest ecologists need to be able to access and interpret LIDAR-derived datasets such as canopy height models (CHM), digital elevation models (DEM), and above-ground biomass (AGB). However, processing this environmental data and visualizing it is a non-trivial task. To remedy this problem, the team has been tasked to create an Android application that enables ecologists to view processed datasets while conducting fieldwork. The app will display this data on a map and cover additional requirements, such as the ability to filter the map data, download limited sets for offline use, click on a location and get a data value for that point, and support both English and French languages. Finally, the map data to be displayed requires a significant amount of storage. So, the team has also set up a server for housing this data.

It is crucially important that the Android app not only meets all the requirements without errors, but that it does so effectively and such that it's easy to use. Testing must be done to check if this is the case. "Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do." [2]. It is vitally important for software development as it allows for needed refinements, such as fixing any bugs in the codebase, ensuring performance requirements are met, and getting feedback from users. For the Biomapper project, software testing will yield unique insights about any problems with the app, enabling the team to make significant improvements. The team has laid out a plan for unit, integration, and usability testing. Unit testing is a way of ensuring that the smallest units of code that can be isolated function as they're intended to. Integration testing involves checking that the different technologies and modules interact appropriately. Finally, usability testing involves testing the app on real users. It is a way of getting feedback from users to ensure that the product is easy to understand and navigate.

These various types of software testing will be carried out in ways that are specific to the project. Unit testing will vary greatly between the Android app and server. Because the Android app has a large codebase, this document will only describe the testing of a small portion of the app. More specifically, testing the key requirements covered by the Base Map class will be discussed, as those units are the most critical to the app's functionality. These include retrieving map data, data filtering, and setting the region of interest. In regards to the server, its ability to serve static and filtered map tiles will be covered. This is done because serving map tiles is the module's primary functionality. Integration testing will cover some of the key interactions between the Android app and server, as well as between their subcomponents. The primary focus is the Android app requesting and receiving data from the server. This must be tested on both ends as it ensures that the app can display map tiles and download them for later use. The results of unit and integration testing will inform the team about any underperforming or broken components, allowing them to make the appropriate improvements and fixes. Additionally, the frameworks used for these types of testing will automate the testing process. This saves time and prevents directly modifying source code to conduct tests. Finally, usability testing will be accomplished by having select groups of users test the Android app. They will be given a list of tasks to complete and a number of survey questions to be answered about the tasks and the app in general. Their feedback will be taken into consideration so the app can be refined further. The initial testing groups are part of research teams working in fields that are relevant to the Biomapper project. This will yield the team more useful responses, and allow them to better tailor the Android app to the needs of the target user base.

Unit Testing

Introduction

Unit testing is a software engineering technique that involves testing an individual unit or component of a software application. The goal of it is to validate the accuracy of a section of code. Another benefit of this technique is it allows software engineers to debug their code. If a piece of code has been unit tested then the process of determining where the error is becomes easier.

In this section of the document, the team will be looking at specific sections of both the server component and Android component. The main functionality that will be tested on the server is the Python filtering script. This Python script is what colors the different values of data. There will be several tests of functionality conducted on the Android side, as it is more extensive in comparison to the server. The team is primarily focusing on components that provide key functionality. Components such as the AboutPage, Preferences, and ActionMenu will not be tested.

For unit testing of the Android application, the team has decided to use JUnit, which is a unit testing framework written in the Java programming language. It is appropriate since the app is written in the Java programming language and relies on the Gradle build automation tool, both of which JUnit is designed for. This framework will be used to test **select** methods in the BaseMap file such as getBaseTileUrlString, getFilteredTileUrlString, RoiMarker, and setFilterValues to show validation of their accuracy. The rest of this section will be broken down into those methods.

Server unit testing will be carried out using the Jasmine framework and a tool called JS-ImageDiff. Built on JavaScript, these are compatible with the team's Node JS. They can automate testing the team needs to prove data integrity. JS-ImageDiff is specific to the primary functionality of the server, which is providing images of map data to the Android app

Server

Outlined below is the overview of how testing will be performed on the server. It will be testing the accuracy of the processed map tiles. The following tests will compare values returned from the Python image filter to a known good image. Known good images will be validated by comparing data values from the original geotiff image to the tiled map images.

1. Data type (ABG, CHM, DEM) API tile retrieval

Description: These functions control where the API retrieves the tiles that are stored in their corresponding directories. By testing these functions to retrieve an expected tile and comparing it to the normal base tile we will know if the controllers are working correctly.
Boundary Values: These functions are called by the application with parameters for data type, zoom level, X and Y coordinates, and lower filter and upper filter value. Based on these values the API knows what function to call, what filter values to apply, and where the file is located.
Example Values: IP:3000/base-tiles/DATATYPE/ZOOM/X/Y/LowFilter/HighFilter Returns a tile of the specified datatype, zoom level, x and y coordinates, and applied the color value filter.
Expected Behavior: Once the call to the API has been made, it retrieves the required tile and runs the python script to edit it. On completion it sends the file to the device to be displayed in the correct position.

2. Filtered tile accuracy testing
Description: To test the accuracy of our filtering algorithm we will be running multiple tests that compare the result of our filtered images vs the actual data values from the original raster image. Using JS-ImageDiff
Boundary Values: The framework we have chosen to use for this test takes in 3 parameters, the original image file, filtered file, and an optional tolerance that can be specified. Returning a Boolean value if both images match within tolerance, or false with an optional .png file which visualizes the differences between the 2 files. This can be useful for troubleshooting bugs in the algorithm.
Example Values: imagediff equal [-t optional tolerance] original.png filtered.png

Expected Behavior:
The team is expecting our data values are accurate with the testing we've done along the way. In order to meet our clients expectations we want to validate these assumptions with a variety of different filter levels applied and checked against original raster values and our tiled images.

Examples of what will be tested below.

2.1: Unfiltered Tile Example

Input Values:

- **Base tile**
- **Filtered tile**

Description:
This case is to test a tile going through the conversion process, but not having any data values altered, will still be the same once the conversion is completed. It is expected to have a 100% match between the base tile and filtered tile.

2.2: Fully filtered

Input Values:

- **256x256 blank png**
- **Fully filtered tile**

Description:
The intention with this test is to discover if any color value is left out when attempting to remove all color ranges from the image. This could occur if changes to the color scale are made but not represented in the code.

2.3: Half filtered

Input Values:

- **Known good filtered png**
- **Filtered tile**
- **Tolerance**

Description:
Testing the accuracy of the Python script by filtering the lower or upper half of the color scale and comparing it to a known good. A tolerance can be specified using the framework which would allow for inaccuracies due to changes in resolution compared to the original geotiff data. Providing a lower number to prove accuracy is preferred.

Android

This section will be broken down into methods that correlate to the BaseMap file. In each method, it will cover the test objective and the various test cases to show validation. The actual programming of the test cases will happen in designated test classes related to the appropriate functions. The content below takes more of a big picture/pseudocode approach in terms of how the testing implementation will go for the various test cases.

BaseURLTesting Test (getBaseTileUrlString)

Objective: Validate that the returned string will return a valid png file

1. Function that retrieves the correct unmodified tile
Description: This function is what makes it possible for the application to display data tiles on the map. It works with the three data types such as DEM, CHM, AGB. The return value of it is an unmodified map tile (base tiles). In order to test this function, the team must ensure that the values given as input do correlate with files on the server. This is a rather easy task to test as the function will either return a base tile or won't return anything at all.
Boundary Values: The function requires four pieces of information to work. It is expecting a data type that can be either CHM, DEM, or AGB. A zoom level that is in the range from 5 to 10 (inclusive). The x values and y values are valid file paths on the server.
Example Values: Function getBaseTileUrlString(dataTypeUrlString, zoom, xVal, yVal) This function is what grabs a base tile from the server to be displayed on the application. It will take in these parameters and make a formatted string in the format of 'dataTypeURLString/zoom/xVal/yVal' This string will then be returned to the calling function which will use a built-in Android function to make the HTTP request.
Expected Behavior: If the parameters match the criteria for each respective data type, then an unmodified tile will be returned back to the application. Failure to enter the correct information such as the wrong zoom level will result in an error message of "Invalid URL."

FilterURLTesting Test (getFilteredTileUrlString)

Objective: Validate that the returned string will return a filtered tile

2. Function that receives the correct modified tile

Description: This function is what allows the key functionality of allowing the user to get specific data for the various data types (CHM,DEM,AGB) depending on a range of values. The return value of this function is a modified tile with the corresponding data values. In order to test this function, the team will ensure that min and max values correlate to the correct data type.
Boundary Values: The function requires six pieces of information to work. It is expecting a data type that can either be CHM, DEM, or AGB. A zoom level that is in range from 5 to 10 (inclusive). The x values and y values are valid files on the server. The min values and max values vary depending on what the datatype is. For example, CHM is expecting a range of 0 to 45 meters, DEM is expecting 0 to 1500 meters, AGB is expecting 0 to 450 Mg C/ha.
Example Values: Function getFilteredTileUrlString(dataTypeUrlString, zoom, xVal, yVal, minVal, maxVal) This function is what grabs a modified tile from the server to be displayed on the application. It will take in these parameters and modify the base tile with the corresponding xVal and yVal. The string returned will be a modified tile.
Expected Behavior: If the parameters match a correct range for the minVal and maxVal , then a modified tile will be returned back to the application. Failure to enter the correct information such as the wrong zoom level will result in an error message of “Invalid URL.”

RoiMarker Test (addRoiMarker)

Objective: Validate that an ROI Marker is created given the conditions

3. Function that creates an ROI marker
Description: This function is what allows the user to create an ROI marker when they open the application. This setting must be set in the previous session for it to take place the next time the user opens the app. In order to test the function, the team will use the conditions of whether this setting is enabled or not.
Boundary Values: The function requires only one piece of information to work. It is expecting a boolean data type that will either be true or false. True will indicate that the user has set the setting of ROI marker to on. False indicates that there will not be an ROI marker.
Example Values:

Function RoiMarker(boolean) This function sets a marker of ROI when the user has turned the setting on for this, which means at this state the boolean is true. Otherwise, this setting is not turned on (boolean is false) and the ROI marker will be there upon the next session of the app
Expected Behavior: Sets a marker of ROI when the user enables that setting.

SetFilters Test (setFilterValues)

Objective: Validate that the correct filter values have been set by Preferences

4. Function that sets filters
Description: This function is what allows the filtering functionality for the user if they are only concerned with one data type at a specific range. To test this function, the team will ensure that the values provided fall in the respective range for that datatype.
Boundary Values: This function requires three pieces of information to work. It is expecting a minKey and maxKey with its respective data type.
Example Values: Function setFilters(minKey, maxKey, dataType) Checks if the minKey and maxKey fall into the correct range for the data type. If not, it returns an “invalid range” message.
Expected Behavior: Allows the map to be filtered on what the user desires for the data type.

Summary

Within this section, only a subset of methods from the BaseMap Java class had been tested. The team felt that this was a sufficient subset to demonstrate the thought process behind the unit testing. As stated earlier, the Android side of the project is more extensive compared to the server. This means that the team is unable to provide more unit testing in this document when it comes to the Android side. With the server's main purpose being to serve static and filtered map tiles to the server, those functionalities have been covered by the tests described above.

Integration Testing

Introduction

In addition to unit testing, the Integration test is another test type that seems really similar to Unit testing. Integration tests combine many different modules and functions in a group to do the test and find out if the interfaces between them are correctly implemented and have the correct response on it. Typical software will contain many different core functionalities and core modules, and normally it will be implemented by different programmers. As a result, it is possible to have errors when they interact and function as a whole. As for the integration testing aspect, the main target of it is to test if the interaction between different software modules/functions is correctly communicating or not. Integration Testing focuses on testing the data communication among core modules and how different components of the Application work together. If we use the car to demonstrate the integration test on it, the integration test if the engine, steering wheels, wheels, and brakes can work together properly. If the engine is working, but there is an error to connect to the steering wheel that the car can not turn right or left, if the engine is working but has an error to connect to the wheels that the car can not move or reverse. To make the car move as we expected, we need to make every individual functionality work seamlessly. And it is the same way in the Biomapper mobile application as well.

The Integration test is necessary for software which has the following significant reasons:

1. Normally, the software will be designed and implemented by different programmers, which leads to different logic and understanding of a function. Integration testing is the way to test if different modules can work together without bugs and errors.
2. There has the possibility that Android Application and server's interface could be erroneous
3. Android Application's different core functionalities' interactions could be erroneous
4. Inadequate exception resolution handling has the possibility to cause issues.

In particular, the team needs their map tile storage, server script, HTTP, API, and In-App functionalities to all communicate and interact with one another effectively. If every part of the application can work together perfectly as expected, the team can achieve the target result that the team's clients want.

Back-End: Map Tiles Retrieval and Filtering

Integration tests mainly cover the interaction between the Frond-End Android application and the Back-end Server. The key of the Back-end integration test involves receiving the request from the front-end then returning map tiles from the file system. This interaction will be done by HTTP requests and API calls from the application to the server.

Integration Test Objectives: Get the unfiltered map tile from the server through an HTTP request.
Test Case Description: A request will be generated that requests the server to get the target map tile from the server file system.
Expect Result: This test will pass the request to the server and get the map tile without a return error or AccessDenied.

Integration Test Objectives: Get the filtered map tile from the server through an API call.
Test Case Description: An API call will be generated based on input filtered data value that requests the server to run the filtering python script through Android's API call.
Expect Result: If the action is performed correctly and the interaction and python script is well working then it should return the filtered map tile to the Android Application.

Front-End: Online Map Viewing and Filtering

Biomapper's front-end integration tests encompass the interaction while sending a request to the server then getting the correct response from it. The key of the front-end integration test involves sending the HTTP requests and API calls to the server and receiving the returned map tiles for viewing or downloading.

Integration Test Objectives: Achieve selecting the target map type tiles and viewing them on the application interface.
Test Case Description: An HTTP request will be generated to request the target map type among four different map types: Normal map, AGB, CHM, and DEM from the server. The map type will be able to select on the application side's testing function.
Expect Result: The target type of map can be viewed on the application interface without errors and bugs.

Integration Test Objectives: Achieve set the filtered map data value base on the selected map type and view it on the application interface.

Test Case Description: An API call will be generated to request the filtered map tiles from the server. The server will run the filtering script base on the input filter value from the Application side then return the map tile asynchronously.
Expect Result: The filtered map tile will be returned to the Android application side and able to view it on the viewing map interface.

Front-End: Download Filtered/Normal Map Tiles Based on ROI

The download filtered/normal map tiles are based on functional interactions between the front-end's set ROI module and Download Manager module. The good interaction between both and server-side can make this integration test performed successfully. The ROI module has a unique algorithm that is able to set the target map tile groups base on input radius. The Download Manager module has asynchronous download functionality to save the ROI area's map tiles to the DATATYPE/ZOOMLEVEL/X/Y folder.

Integration Test Objectives: Download the ROI's map tiles group asynchronously to internal storage.
Test Case Description: The ROI manager will generate a group of target map tiles based on input radius value and coordinates. The map tiles group will be downloaded by the download manager then saved by sequence to the internal storage folder.
Expect Result: All map tiles will cover the ROI's area and save them to the internal storage target folder without errors and bugs. Then the offline map viewing function is able to access and view the map from the DATATYPE/ZOOMLEVEL/X/Y folder when the offline mode switch is on.

Plan

Integration testing will be carried out in the coming weeks to ensure all modules connect as they're intended to. The tests described above cover the key relations between the primary components of the project. There are still some stretch goals that have high priority to be implemented. As these are added, any newly established interactions between modules will be tested.

Usability Testing

Introduction

Usability testing plays an important role in software development. It involves testing an application with a target audience who understands or is interested in the topic. The benefit of usability testing comes from the feedback provided by the user. Giving an unbiased look at how an application performs and the overall simplicity of its design.

This section of the document aims to outline the team's plan regarding usability testing Acquiring a group of relevant testers with experience relative to the application will provide more useful feedback. This can include feedback on performance, UI/UX (User Interface/User Experience) design, areas for improvement, and uncover potential bugs.

Plan

Over the next week our team will be conducting usability testing, giving us enough time to implement any final recommendations. Chris Doughty, the team's client, will be providing us with a group of around 8 researchers from his team at Megabiota Lab and the potential of more testers from Dr. Goetz's team Global Earth Observation & Dynamics of Ecosystems Lab. [4] This provides a pool of users who will understand geospatial data, the complications that come with it, and the potential usefulness for the application. Tests will be conducted either remotely over Zoom meetings, if a tester has an Android device, or scheduling an in-person session for a device to be provided to the tester. A list of tasks will be provided to the user with a section for feedback on ease of use, overall usefulness, and comment section for improvements to make. Completion of tasks does not have to be in order and should be part of naturally exploring the application. If clarification is needed, the team will be present to respond to any questions and should note commonly asked questions. Once a tester completes all tasks, their answers are reviewed, and a set of follow-up questions can be asked to get more in-depth feedback. After all user tests are completed, the team will compile the results and determine which the suggested changes should be implemented and how this will be done.

Task list for testers

Task	Simplicity (1-10)	Ease of Navigation (1-10)	Usefulness (1-10)	Comments on improvements we can make
Try to display a new data type.				
Try to apply a data filter to your current data type.				
Try and select a Region of Interest				
Try to download your Region of Interest				
Turn Offline Mode on and disconnect the device from the internet.				
Try to find the value for a location on the map.				
Other comments:				

Follow up questions:

- Please describe how you feel about the applications performance.
- What did you find most difficult to understand or use?
- Do you have any suggestions on what should be changed?
- How useful do you believe this application will be for in field research?
- What would you add to the app to improve its usefulness?

Schedule

Over the next week, the team's user testing plan will be executed and its results will further assist with the app refinement process. First, the lab group that our client is a part of (the Megabiota Lab) will be provided a copy of our app and the task list and questions provided above. Then, the app and questions will be distributed to Dr. Goetz's group (the Global Earth Observation & Dynamics of Ecosystems Lab). Once the team has received feedback from all users, their evaluation will be considered when refining the Android application.

Conclusion

Tropical forests are vitally important to all of Earth's ecosystems and the species they contain. The ecologists that work within them are partially responsible for their protection and preservation. But, ecologists struggle with accessing and interpreting the datasets they need for their work. What would greatly help forest ecologists is an Android application that displays these datasets on a map and includes additional useful tools, such as data filtering and downloading. This, along with a server for storing the map data, is what has been created as part of project Biomapper.

All that remains to be done is software testing to verify that it meets all the requirements and so refinements can be made. This document outlined the plan for conducting this testing and discussed unit, integration, and usability testing. Testing the Android application's codebase will be done using the framework JUnit. The server will be covered by the framework Jasmine and an additional tool known as JS-ImageDiff. Finally, the product as a whole will be tested by a select group of users that are knowledgeable in fields relevant to the project.

These tests will be carried out in the imminent future and the team and their clients are excited to have a finished product.

References

[1] “Why are rainforests important?”. Rainforest Concern.
https://www.rainforestconcern.org/forest-facts/why-are-rainforests-important.

[2] “How does software testing work?”. IBM.
https://www.ibm.com/topics/software-testing

[3] “Integration Testing: What is, Types, Top Down & Bottom Up Example”. Guru99.
https://www.guru99.com/integration-testing.html

[4] “Global Earth Observation & Dynamics of Ecosystems Lab”. NAU.
https://goetzlab.rc.nau.edu/