Software Testing Document: Version 1.0

Lunar Pit Patrol

Levi Watlington, Alden Smith, Caden Tedeschi, and Evan Palmisano

Sponsor: Trent Hare

Mentor: Vahid Nikoonejad Fard

11/05/2024

Table of Contents

Introduction **3**
Unit Testing **4**
- Goals of Unit Testing 4
- Unit Testing Approach 5
- Testing Units 6
- Takeaways 8

Integration Testing **8**
- Integration Testing Goals 9
- Integration Testing Approach 9
- Integration Testing Plan 10

Usability Testing **11**
- Usability Testing Goals 12
- Usability Testing Plan 12
- Usability Testing Methods 13
- Data Recording and Analysis 13

Conclusion **14**

Introduction

CraterStats is a program designed to generate graphs based on statistical computation of crater data. Currently, its command-line interface poses challenges for users, particularly those unfamiliar with terminal commands or command-line operations. Our project addresses these issues by developing an intuitive graphical user interface (GUI) that enhances navigation, usability, and user experience. To achieve this, we incorporate Python libraries for graphing and UI design, ensuring the application meets several key requirements: an easy-to-use interface for non-technical users, accurate graph representation, saving user-generated plot configuration files, and exporting generated plots.

Software testing is a critical step of the software development lifecycle, serving to ensure that applications function as intended, are free of defects, and provide a reliable user experience. The motivations behind software testing include the need to verify functionality, enhance performance, and improve user satisfaction. Effective testing not only enables us to identify bugs and issues before the application reaches the end user but also gives confidence in the reliability and quality of the software the team has developed.

This document outlines our testing strategy for the CraterStats GUI interface, which features various types of testing, including unit testing, integration testing, and usability testing. We plan to conduct thorough unit tests on individual components of the interface to ensure they function correctly in isolation. Integration testing will focus on the interactions between components, particularly how user inputs translate into commands for generating graphs. Usability testing will involve gathering feedback from actual users to confirm that the interface meets their needs and expectations.

The logic and reason behind our testing plan are rooted in our application and its intended user base. Given that CraterStats targets users, specifically astrogeologists who may need to be more technically inclined, it is crucial to prioritize usability and ensure that the interface consistently performs as intended. We have opted for testing in

areas directly impacting user experience, such as interface responsiveness and the accuracy of graph displays to prevent potential frustrations that could arise from any failure of these functionalities.

Unit Testing

Unit testing is a critical aspect of software development, allowing developers to detect issues early in the development cycle. It is a method of testing designed to assess specific components of a software application. These components can be any specific parts of a program from functions to classes. Each unit is the smallest piece of software that can be logically isolated and tested. When a unit test is conducted, specific features and functionalities are isolated from the program to ensure that they work as expected. With unit testing, we can ensure code reliability by testing each unit separately allowing the team to catch bugs early. Managing code maintenance allows unit testing to act as a safeguard against bugs that are introduced when code changes or new features are added. Finally, unit tests also serve as documentation, showing how a function or class is intended to be used and what output is expected for given inputs. When implementing these tests, we want them to be isolated from the rest of the application, repeatable with the same output every time the test is conducted, and fast thanks to the small size of the units.

Goals of Unit Testing

Our team has high expectations for application testing to ensure functionality and usability. For a GUI interface, it’s important to validate that the background code performs as intended while also guaranteeing that all user inputs and interactions function smoothly, reliably, and consistently. This involves not only unit testing to confirm that individual functions work in isolation but also integration and UI testing to verify that all components interact correctly and present an uninterrupted experience to users. With thorough testing, our goal is to maintain quality, reduce bugs, and provide a reliable, user-friendly interface.

Unit Testing Approach

Our approach to unit testing involves using Python's unit testing libraries, such as **unittest** and **pytest**, to streamline our testing activities. We also plan to determine test-related metrics like test coverage, focusing on line coverage to measure how much of our code is executed during testing. We aim to test key input options to ensure their values match stored values accurately. For instance, we will test that when a user selects the "Cumulative" radio button, the program correctly stores "cumulative" in its plot configuration. This process involves implementing assertion testing to verify that the inputs and their corresponding stored values are accurate.

While we plan to conduct detailed unit testing for these critical methods and procedures, our focus will primarily be on the functionalities that directly impact the user experience. By prioritizing these elements, we ensure that the most crucial parts of our system are robust and reliable, while also laying a solid foundation for comprehensive testing in future development phases. The units we will be testing are UI elements such as radio buttons, dropdown menus, check boxes, and input boxes.

Each unit test is designed to run by itself, allowing us to isolate failures and diagnose issues easily and quickly. Python testing libraries enable us to run these tests repeatedly, especially after any updates or changes to code within the application, ensuring that each component remains functional and that the new code introduced to the application does not interfere with features we have already established.

For each of these components, we develop tests that handle both typical user behavior and outlying situations. As an example, text inputs are tested not only with normal valid entries from a user but also with unexpected inputs, such as special characters or empty fields to check that the application processes these inputs appropriately without breaking. Dropdowns and radio buttons are tested to verify they return correct default values if a user does not make a choice, showing that the application can function predictably even if some settings remain unchanged

Testing Units

Testing Global Settings

When the application opens, users are greeted with the Global Settings page, an important area where they configure settings that influence the application's output. To ensure reliability, we focus heavily on unit testing the interactive elements—dropdown menus, radio buttons, checkboxes, and text inputs—to verify that every input captures and processes user choices accurately.

Our unit testing process for these elements is similar for every unit. Using our Python testing libraries, each component is isolated and tested individually to confirm that it performs as expected in response to user actions. For instance, dropdown menus are tested to ensure they display the correct options and return the selected value accurately. Similarly, radio buttons are tested to verify they allow only one selection, capturing and storing the user's choice effectively. Checkboxes are tested to make sure they return both true (checked) and false (unchecked) values consistently, and text inputs are tested to ensure they accept and verify the entered data correctly within any preset constraints.

Testing Plot Settings

The Plot Settings page, much like the Global Settings page, contains various inputs, including text boxes, dropdown menus, and checkboxes. For each input type, we conduct unit testing to ensure each component works as intended in isolation. Given that this page contains a large number of text boxes for setting plot details like Title, Subtitle, and subplot name, we validate that each text input processes the data accurately and adheres to any defined constraints. For example, plot font size and plot scaling inputs are tested for valid entries, ensuring that numbers within acceptable ranges are processed correctly. We also account for edge cases by testing invalid entries, such as overly large values or incorrect data types, to confirm that the application handles them.

In addition to typical UI elements, the Plot Settings page includes a data file upload feature, which presents unique challenges for unit testing. We validate that the file upload input only accepts data in the correct format, and we test for cases where an incorrect format is provided. Furthermore, we simulate various scenarios, such as corrupted or poorly formatted data within an otherwise valid file, to confirm that the application can identify and handle these cases. Testing the file upload functionality in isolation allows us to catch potential errors before data processing, ensuring that files are read and parsed accurately without affecting the rest of the application.

Toolbar Testing

The toolbar contains features that complement the user's progression with the application. The toolbar will be tested using the same standards as the global and plot settings. We will test every part of the toolbar to see if they work as expected. Much like the other settings tabs, we will be using Python libraries to test the toolbar options.

This toolbar consists of several options, one of which is the file menu. This menu consists of several options: save, open, and close. We will test this by isolating all of these options and verifying that they work. For the save, we will check if the data that is saved is the same as the data being stored. For the open, we will check if a file is opened and read using assertions. As for the close option, we will check for proper application termination.

Another toolbar feature is the plot option. It consists of new, duplicate, and delete. We will test this similarly. For the new option, we will test if a new plot was created. For duplicate, we will test if a plot was duplicated successfully using assertions. Finally, for delete, we will test if a plot was successfully deleted. The third option is the export feature. It consists of several options too: image, summary table, and .stat table. To test these, we will check if an image is exported successfully and test if a summary or stat table is created successfully, concerning their respective options.

The final feature on the toolbar is utilities. This feature is comprised of demo and dev notes. To test these, we check if the demo is running successfully, and if the dev notes are showing, with respect to their options. That is to say demo will be tested to see if it creates demo plots and starts a demonstration, and if the dev notes option shows the developer notes.

Takeaways

Our unit testing ensures that these components function consistently across various scenarios, providing the user with a reliable interface for modifying settings. Through these series of tests, our goal is to deliver an application where each interactive element performs its role within the overall user experience. Through this thorough testing, we will be able to detect and address more bugs and issues present in the code. More specifically, we will be able to spot any usability issues that may negatively impact user experience. Thus, we need to commit to this testing plan.

Integration Testing

Integration testing is a critical stage in the software development lifecycle, aiming to ensure that individual modules or components work together as intended. While unit testing verifies the functionality of isolated components, integration testing focuses on the interactions between these components, ensuring data is passed correctly and functionality across boundaries remains intact. In the context of the Craterstats Capstone project, integration testing verifies that the different modules of the graphical user interface (GUI), back-end processes, and external dependencies (such as the Craterstats CLI and underlying database) interact as expected. By conducting these tests, we aim to identify and resolve any issues that may arise from the “plumbing” or connections between modules.

Our approach to integration testing involves selecting key integration points and designing test cases to evaluate data flow, module interactions, and correct

implementation of input-output contracts. This approach ensures that major interactions, particularly those influencing core functionality, are thoroughly validated. We aim to create a robust and error-free interface by confirming that modules work cohesively, whether they involve user input processing, database queries, or graphical plotting.

Integration Testing Goals

The primary objectives of integration testing for our Craterstats GUI project are:

1. Validate Module Interactions: Ensure that major modules, such as the GUI, data processing, and plotting modules, interact correctly and fulfill their intended roles in response to user inputs.
2. Verify Data Exchange: Confirm that data passed between components is accurately formatted and contains the necessary information for downstream modules.
3. Check Module Boundaries: Test the boundaries where different modules connect, including the handling of external libraries (e.g., CLI) and internal processing (e.g., configuration files, plot generation)
4. Ensure Robustness: Identify potential points of failure in interactions, allowing us to resolve issues before they affect overall performance and user experience.

Integration Testing Approach

To structure our integration testing, we focused on the boundaries where modules interact in significant ways. The following steps were used to define and implement our testing approach:

1. Identification of Key Integration Points: We examined the craterstats GUI architecture and identified modules where critical data exchange and function calls occur. These include:
    a. The GUI-to-CLI interface, where user input from the GUI is passed to the craterstats CLI library for processing

    b. The Data Processing and Plotting Modules, where data retrieved from the CLI library is visualized in plot form on the GUI
2. Test Harness Design: For each integration point, we designed specific test harnesses to simulate real usage from users and verify the accuracy of data transfer. For instance:
    a. The GUI-to-CLI interface was tested by setting up mock inputs from the GUI and verifying that corresponding commands were passed to the CLI library correctly.
    b. The data processing module was tested by providing various data inputs and observing whether the plots generated matched expected outputs.
3. Validation of Assumptions: Each test harness was configured to validate the assumptions expected by each module. For example:
    a. Data format verification: Tests ensured that the data structures expected by the plotting module matched the output format from the processing module.
    b. Boundary testing: Tests were created to validate the accuracy of interactions under varying input sizes and types, including edge cases (e.g., invalid user inputs, large data files).
    c. Error handling: We simulated erroneous inputs and confirmed that modules managed errors gracefully, returning helpful feedback without disrupting the GUI.

Integration Testing Plan

For each integration point, we applied the following test cases and verification methods:

1. GUI-to-CLI interface testing
    a. Test Harness: A mock GUI input sequence was created to simulate user commands, replicating different inputs such as data range selections and plot type selections
    b. Verification: We checked the output by intercepting and analyzing the data passed to the CLI module, confirming command consistency and format.

    c. Expected Outcome: The CLI receives and processes commands without data loss or misinterpretation, and any resulting outputs align with GUI expectations.
2. Data processing and plotting module testing
    a. Test Harness: Input files containing crater data and specific configurations were prepared and passed to the data processing module.
    b. Verification: The output plots were visually inspected and programmatically compared against reference images. Automated checks validated that plot attributes such as labels, markers, and scaling matched the data provided.
    c. Expected Outcome: Generated plots should consistently reflect the input data, with no visual inconsistencies when settings are modified.

By implementing this integration testing plan, we aim to ensure that the craterstats GUI provides a cohesive and reliable user experience. The focus on integration boundaries and careful validation of data exchange will help us detect and resolve issues that could impact functionality. This approach strengthens the overall quality of the program, ensuring that the system behaves as intended in real scenarios and meets the expected standards of accuracy and reliability for crater mapping applications.

Usability Testing

Usability testing is a critical component in the development of user-facing applications. Its primary goal is to evaluate the effectiveness, efficiency, and satisfaction with which end-users can access and utilize the functionalities provided by a software system. By focusing on the interactions between users and the application, usability testing ensures that the software is not only functional but also intuitive and accessible to its target audience.

The objectives of usability testing are multifaceted. Primarily, it seeks to identify any potential barriers that users may encounter while navigating the application. This involves examining the overall quality and understandability of the user interface, the logical flow of the application, and the ease with which users can accomplish their tasks. For applications like CraterStats, where the user base may not be particularly patient or technologically savvy, effective usability testing becomes even more essential. It is vital that users can seamlessly interact with the software, as any difficulty may lead to frustration, reduced productivity, or a reluctance to use the application altogether.

In general, usability testing involves several steps: planning the tests, recruiting participants, conducting the tests, and analyzing the results. This process can take various forms, including user observations, interviews, focus groups, and task-based evaluations. By leveraging these methods, developers can gain valuable insights into user behavior and preferences, informing necessary adjustments to the application's design and functionality.

Usability Testing Goals

Our usability testing will focus on four primary goals that align with the typical actions users are likely to perform while interacting with the CraterStats interface:

1. **Finding and Toggling the Demo**: Assessing how easily users can locate and use the demo feature of the application.
2. **Recreating a Sample File**: Evaluating the ability of users to accurately recreate a sample file using the provided tools.
3. **Saving and Locating Exports**: Testing the efficiency with which users can save plots and subsequently find those exported files.
4. **Creating Custom Plots**: Observing how well users can generate custom plots without specific guidelines, testing their understanding of the application's features.

These goals encompass essential interactions that users will likely engage in frequently, making it imperative to ensure that the interface supports these tasks effectively.

Usability Testing Plan

The usability testing plan for the CraterStats interface is informed by several key considerations related to the nature of our application and its intended users. The target audience primarily consists of astrogeologists who may lack extensive experience with command-line interfaces (CLIs) or complex software systems. This lack of technical expertise underscores the importance of designing a user-friendly interface that facilitates intuitive interactions. The novelty of the product—transforming a CLI application into a graphical user interface (GUI)—also heightens the need for comprehensive usability testing, as users may have different expectations and behaviors when engaging with the new interface.

Moreover, the consequences of poor design can be significant, impacting users' ability to generate accurate and timely data visualizations crucial for their work. Therefore, we must prioritize testing areas that directly influence user satisfaction, workflow efficiency, and overall application reliability.

Usability Testing Methods

To achieve our testing objectives, we will employ a mix of quantitative and qualitative methods:

1. **USGS Outsourcing**: We will gather qualitative feedback by allowing a select group of astrogeologists partnered with USGS to download and use our application. They will provide their experiences with the interface, allowing us to explore user perceptions, expectations, and areas of confusion.
2. **User Studies**: We will perform task-based user studies where participants will be asked to complete specific actions within the application, such as toggling the demo and recreating a sample file. We will ask for recordings of these tests using screen capture software or Zoom and note any difficulties or hesitations encountered by users.
3. **Sponsor Reviews**: We will engage our sponsor to review the interface and provide feedback based on established usability principles. These reviews will occur before the release of our Alpha version to identify usability issues throughout the design process.
4. **Acceptance Testing**: Finally, if possible, we will conduct acceptance testing with a larger group of potential end-users to validate that the application meets their needs. This phase will involve as many participants as we can accrue and will focus on whether users feel the application is intuitive and meets their expectations.

Data Recording and Analysis

To ensure the effectiveness of our usability testing, we will implement a comprehensive data collection and analysis strategy. Each testing session will be recorded, and feedback will be documented through observation notes, participant interviews, and post-task surveys. We will analyze the data to identify patterns, common issues, and areas for improvement.

By adopting this thoughtful and well-structured usability testing plan, we aim to ensure that the CraterStats interface meets the needs of its users, ultimately leading to a successful application

launch. The findings from our testing will guide further refinements, enhancing user experience and satisfaction while helping astrogeologists achieve their research goals efficiently.

Conclusion

In conclusion, the development of the CraterStats GUI represents a step toward improving the accessibility and usability of craterstats for astrogeologists. By transitioning from a command-line interface to a more intuitive GUI, we aim to eliminate issues faced by users who may not be technically inclined, improving overall user experience and productivity.

Through unit testing, we will ensure that individual components function correctly in isolation, while integration testing will validate the interaction between these components. By focusing on integration points, we aim to maintain data integrity and enhance the system's overall reliability. The integration testing plan is designed to identify potential points of failure and resolve them proactively, ensuring that all modules work cohesively and effectively to support user interactions.

Equally important, our usability testing plan entails the need for a user-centered approach. By gathering feedback from actual users, including user studies and sponsor reviews, we will obtain valuable information about user behavior and preferences. Our usability testing reflects our understanding that an intuitive interface is necessary for improving user satisfaction and maintaining the effective generation of accurate data visualizations.

With these testing methodologies implemented, the CraterStats GUI will emerge as a functional and usable software product. The functions of focus in both integration and usability testing guarantees that we can deliver an application that not only meets but exceeds user expectations while providing reliability. By acknowledging the needs of our users throughout the development process, we provide the CraterStats GUI as a valuable tool for astrogeologists, supporting their research endeavors and enhancing their ability to generate meaningful insights from crater data.

Notice

Elements of this document were developed with the assistance of Artificial Intelligence. All testing topics are original and authentic ideas developed by the Lunar Pit Patrol. All conducted tests are/will be generated and conducted exclusively by the Lunar Pit Patrol and USGS Astrogeology.