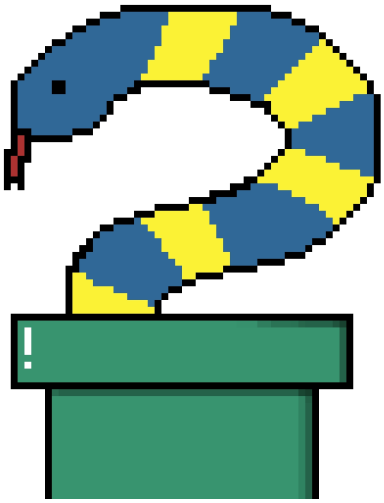

Pypline Software Testing Plan

Version 1.0

2 April 2019

Sponsor	Scott Akins Dr. Jay Laura
Mentor	Isaac Shaffer
Team	Pypline
Team Members	Nicholas Anderson Austin Collins Connor Schwirian Abdulaziz Zarie

1. Introduction

The mission of the United States Geological Survey Astrogeology Center (USGS) is to further our knowledge of the solar system through their research into planetary cartography, geoscience, and remote sensing. Among their responsibilities are the development of a software toolkit for working with planetary images called the Integrated Software for Images and Spectrometers (ISIS), participation in mission planning, and the archive of all NASA planetary image data. USGS maintains the Planetary Image Locator Tool (PILOT) website to facilitate the distribution of planetary image data to the scientific community and the general public. The site allows for the selecting of images from the planetary database and utilizes their in-house processing cluster to apply desired ISIS processing tools onto the images. Our sponsors, Scott Akins (USGS IT Specialist) and Dr. Jay Laura (USGS Research Scientist) have brought Pyline on to upgrade the current Projection on the Web (POW) implementation, which lies between the user selecting images on the PILOT website and processing on USGS's processing cluster.

In order to deliver an error free product, that is up to the USGS standards, a robust software testing plan is required. To cover all aspects of the program design, the testing is separated into three categories: unit, integration, and usability testing. Unit testing will focus on each of the four main program components. As the REST API and generator library are of particular importance to the client, the majority of unit testing will focus on those areas. Given that the Airflow scheduler is third party application put in use for the project, and not developed by Pypline, the unit testing needs will be relatively minimal. Integration testing will verify that the interactions and data exchanges between our separate components are functioning correctly. While all interfaces will be tested, of particular importance and focus will be the generation of DAGs by the generator program. DAGs will contain the statements for execution on the USGS cluster and need to perform as expected to prevent misuse. Containerization of the overall solution was one of USGS key requirements in the project proposal. Because of this, the exchange of data between the containers is another area that needs extensive testing. While the majority of Pypline functions as back end product, usability testing will ensure that the front end website is intuitive for the end user.

The following sections detail the testing plan components and how they will interact to provide complete testing coverage of the new imaging pipeline. This will allow the team to identify and correct any usability regressions during development.

2. Unit Testing

Unit testing is base level of software testing. Its goal is to ensure that each of the individual components in an overall design are performing as expected and to their specifications. Normally this involves testing the individual functions or methods within a component. This will allow any errors of issues at a fundamental level to be identified and corrected. Additionally, unit testing scripts will be run after any changes are committed to ensure that no regressions are introduced during the product development cycle. The main units of our application are UI, REST API, generator library, and the Airflow scheduling container. Each require a different approach to unit testing suited to their purpose.

2.1 UI

The UI is the only user interactive part of the program and provides several key functions: allowing the user to specify images to manipulate in the form of a csv file and manipulating the desired ISIS commands. Parsing the CSV files involves extracting the image URLs and identifying the selected mission from the include instrument tag. Since we are accepting input from the user, we need to ensure that the file is valid and handle any failures gracefully. Given the relatively simply design of the UI, the method of unit testing for the UI will be based on testing scripts that will be performed after any modifications.

Name: Correct CSV File
Purpose: Test image url and mission tag extraction.
Description: CSV files that were generated using the PILOT website for both SSI and MRO are uploaded into the website. Website transitions correctly to recipe modification form and hides the upload dialog box.

Name: Bad CSV File
Purpose: Test UI response to a bad/corrupt CSV file .
Description: A CSV generated without valid USGS image URLs is submitted to the UI. A message to user regarding the incompatibility of the file is returned to the user and site asks for resubmission.

The second essential function of the UI is to parse the command recipe and provide the opportunity for the user to modify it. Several of missions include instrument specific ISIS commands, notably the first to convert the image data from the various mission specific

formats to the ISIS standard CUB format. The UI's ability to correctly retrieve the correct mission will be essential for a successful run. Additionally, the form presented to the user must allow for modifications and notify the user on a successful job submission.

Name: Correct Recipe Retrieval
Purpose: Test that the correct mission recipe is obtained and presented based on CSV instrument tag.
Description: CSV files that were generated using the PILOT website for both SSI and MRO are uploaded into the website. The corresponding mission recipes is presented to the user for both test cameras.

Name: Accepting User Values
Purpose: Default value boxes are allowed to be modified to user specified values.
Description: After CSV upload, all default boxes are able to accept input from the user.

Name: Incorrect Variable Type Entry
Purpose: Test the type checking embedded within the recipe.
Description: After CSV upload, all default boxes are able to accept input from the user. Purposefully entering string in a entry for an numeric value presents an error message when the submit button is pressed.

Name: Allow Early Recipe Termination
Purpose: Tests that recipes can be stopped at any point.
Description: After CSV upload, unchecking a box will cause all lower commands to become unchecked as well.

Name: Output Selection
Purpose: Allow the user to select between image download and file storage.
Description: One of the final premement selections is the handling of output: either for download or file storage. The website correctly show/hides the additional fields required for processing of the output files.

Name: Submission Success Redirection
Purpose: Test the notification to the user on a successful job submission.
Description: After submit button is pressed, the website redirects to a notification page.

The UI component includes several areas that require both correct and misuse testing to ensure that user facing component remains flexible and responsive to the user.

Maintaining successful completions of the UI unit tests will ensure that the data being submitted to the REST API is acceptable.

2.2 REST API

The REST API not only functions as the submission target for our UI but has the ability to accept any future front ends required by USGS. The API needs to be checked that it is adhering to the specifications that we provided and can utilized untethered from the packaged UI. The command line nature of the product lends itself well to sending in test commands to verify the different functions provided. This includes not only a successful response but that it can also generate errors on bad requests. Additionally, the API includes a test function that can be utilized in future development that requires unit testing to remain functional when the final product is delivered to USGS.

Name: Successful Response
Purpose: Test that a successful response from the API can be generated with a valid request.
Description: A valid request is sent to the API from within the FlaskUI container directly to the REST API. A "Response: Generation successful." is received back to the terminal.

Name: Failure Response
Purpose: Test that the API returns a failure message on invalid API request.
Description: A syntactically invalid request is sent to the API from within the FlaskUI container directly to the REST API. A "Response: Generation failed." is received back to the terminal.

Name: Test Response
Purpose: Ensure that the test functionality of the API remains to support future development.
Description: A test API message is sent to the API from within the Flask UI container. A test successful message is returned and the test DAG is produced.

While the REST API works closely with the generator to produce the needed workflow DAGs, several functions within the API need individual testing. Proper testing of the APIs functionality of the API allows any issues within the generator to be identified and isolated within that module.

2.3 Generator

Our generator library provides a singular function: converting a user's request for a job into an executable workflow. This library consists of 2 functions: one that parses and organizes a user's request into a usable format, and another that generates an Airflow DAG to be executed by our scheduling container. These two functions are assisted by two data structures: an object that stores an individual processing command and its parameters as well as an object that stores one of these command objects with additional behavior that allows the command to be represented in a DAG.

Due to the nature of the generator as a Python library, it is easily testable through Python scripting. Testing will consist of simple units that demonstrate the behavior of the 4 components described above.

Name: Command Object Creation
Purpose: Test the creation of a valid command object with correct behavior.
Description: An example ISIS command is used to create a command object and this object is tested by ensuring its string representation matches its expected value.

Name: DAG Object Creation
Purpose: Test the creation of a valid DAG object with correct behavior.
Description: An example command object is used to create a DAG object and this object is tested by ensuring its string representation matches its expected value.

Name: User Request Parsing Success
Purpose: Test the parsing of a valid user request and resulting behavior.
Description: A valid user request, in the form of a JSON object, is passed into the generator library's request parsing function. The result of this function call is then tested against its expected format and content.

Name: User Request Parsing Failure
Purpose: Test the parsing of an invalid user request and resulting behavior.
Description: An invalid user request, in the form of a JSON object containing incomplete data, is passed into the generator library's request parsing function. The result of this function call is then tested to ensure it represents this failed attempt.

Name: DAG Generation Success
Purpose: Test the generation of a DAG from valid, correctly-formatted user request data.
Description: A valid, reformatted user request is passed into the generator library's DAG generator function. The result of this function call, a generated DAG, is then tested to ensure it is a well-formed and executable DAG.

Name: DAG Generation Formatting Failure
Purpose: Test the generation of a DAG from incorrectly-formatted user request data.
Description: A poorly formatted user request is passed into the generator library's DAG generator function. The result of this function call, a representation of a failed DAG generation, is tested.

Name: DAG Generation Data Failure
Purpose: Test the generation of a DAG from invalid, correctly-formatted user request data.
Description: An invalid, reformatted user request is passed into the generator library's DAG generator function. The result of this function call, a representation of a failed DAG generation, is tested.

2.4 Scheduling Container

The Airflow scheduling software that is being deployed as part of Pypline is an large project being incubated by the Apache Foundation. As such, the specific inner workings of the product can largely be excluded from Pypline's unit testing. The areas that do need individual testing are the modifications that we made to integrate the product within our solution. Specifically, the containerization of the product is not a standard use case and requires testing of its Docker build. Additionally, Airflow is generally used to perform monthly, weekly, and daily processing jobs. Several settings modifications had to be implemented to achieve the immediate DAG pickup and execution required by our project and need to be tested for regression testing on any changes. Finally all components need to be tested for failures and the scheduler can potentially be feed an incorrect DAG.

Name: Docker Build
Purpose: Test that the Airflow and ISIS container can be built from the Docker file.
Description: Rebuild the image from the Docker file published to the Github repo. This will ensure that the product can be replicated as needed and any needed requirements

are kept within the build file and not directly applied in the container. The build process completes with any errors generated.

Name: Immediate DAG Pickup
Purpose: Test the configuration of Airflow conforms to Pypline needs.
Description: A known working DAG is placed within the DAG pickup folder in the Airflow container. The DAG is parced and processed by the Airflow scheduler within 15 seconds as indicated by the web interface.

Name: DAG Parsing Failure
Purpose: To ensure that Airflow generates appropriate failure messages on DAG failure.
Description: A known DAG with syntax errors is placed in the DAG pickup folder. The web interface indicates the failure and sends an email to the correct address specified in the DAG to simulate notification to USGS staff.

With the validation of the Airflow server, each part in the total solution will be verified as functional. The next stages of software testing can be performed to ensure that each component works together and the interfaces between function as designed.

3. Integration Testing

The testing plan up to this point has focused on the individual software components. However, correctly performing units do not necessarily ensure that changes have not disrupted the overall functioning of the software and introduce systematic errors. Integration testing focuses on the interactions between the units to verify that they continue to function together throughout the development process. This is especially important for Pypline, as our product is required to be containerized. One of the features and challenges of using multiple containers is the successful exchange of data as seen in the Generator to Airflow interface.

3.1 UI to Generator

The UI needs pass a valid API request to the REST interface which will produce the specified DAG. Testing involves ensuring that the selections made from the UI are preserved throughout the generation process. This includes both the overall commands in addition to the specified parameters. We already tested the individual units handling of of receiving data that contains errors within the units themselves.

Name: User Commands Converted to DAG
Purpose: Verify that the user's selected workflow is represented correctly as a DAG.
Description: Start the Flask UI container by itself and load a CSV files for SSI into the UI. Remove the last two commands and change the default values within the parameter dialog boxes. Examine the generated DAG for the changes specified in the UI.

3.2 Generator to Airflow

Breaking the containerization is important for the overall functionality of our product. Luckily, Docker volumes provide an easy method for passing files between containers. As part of integration testing that linkage needs to be checked to ensure that any container changes have not broken that linkage.

Name: Container DAG Exchange
Purpose: Ensure that the DAGs are correctly passed between the two project Docker volumes.
Description: Start up both container but shutdown the Airflow Server. Generate a DAG as seen in the UI to generator testing. Using the command line in the Airflow container verify that the DAG is present in pickup folder. Start Airflow and verify the DAG is seen within the web interface.

3.3 Airflow to Website

To finish the entire loop, Airflow needs to package up the images so they can be delivered to the user. Final image results need to be packaged into a zip file for download by Airflow. The UI will display a link for download when ready.

Name: Download Link
Purpose: Ensure that the generated files are available for download after processing.
Description: Start both containers with the autostart scripts. Upload a CSV into the UI and choose the default parameters. Ensure that the download option is selected. Wait until the submission page indicates that the download link is ready. Download and unzip the process images.

After integration testing is completed, the products intended design can be validated as functional. However, the usefulness of software cannot always be determined by a functioning design. Actual user input through usability testing can heavily influence changes that need to be made to the software design.

4. Usability Testing

During usability testing real users interact with the application to test its ability to achieve the end goal. The aim is for ease of use across all fronts and is specifically designed to show flaws that don't follow the path of ease. Users will be given specific tasks to perform while being observed by someone such as a developer of the program. Some will be more successful than others but should also highlight areas of repeated increase in a challenging task or confusion. This will give the developers an idea of where they need to focus to linearize the ease of use. Usability testing also removes bias as feedback comes straight from the user on the spot.

To operate and use our pipeline, the user should have basic knowledge of PILOT, ISIS commands, and NASA missions. As the client states, the users of this pipeline will mostly consist of scientists, researchers, high school students doing planetary projects, people interested in interstellar photos, and employees at USGS. The pipeline will handle processing of ISIS commands on pictures submitted by the user. The user chooses the pictures they want to process from the PILOT website and will download the .csv file. Then they will open our user interface in which they will be presented with the specific ISIS commands for picture processing where they choose where to stop the commands. Then the user will submit the pictures for processing once their specifications have been finalized. The user will then be redirected to a thank you page where it shows the processed images in a zip file.

When taking the main use cases into consideration, the user bases we decided to split the focus groups into employees working at USGS and potential non-staff users. When the pipeline is deployed at USGS, the majority of processing jobs will be generated in-house with a smaller portion coming from interested individuals trying the service out. Therefore, when testing the user groups we will focus on how easy it is for users to understand what they need to input. We will also be examining how fast can the UI provide them with the necessary information and any errors that appear in the program while being used. In the following section we will provide more information regarding the testing groups.

Focus Group: Employees at USGS
Plan: We will use Scott Akins as our initial user test and reach out to other USGS employees if granted permission. We will test the user interface usability and ease of access with 2 or more employees at USGS.

Method of recording: The user testing segments will be recorded by a survey at the end of the segment.
How Often: Every time our UI is updated from user and mentor input during our monthly meetings.

Focus Group: Non-staff Users
Plan: We will ask CS students from our capstone class test out the pipeline.
Testing our pipeline interface with 2 or more non-staff tech savvy persons. The testing will be done in sessions where we let the user navigate through the system to achieve their desired image output.
How Often: Every time our UI is updated from user and mentor input.

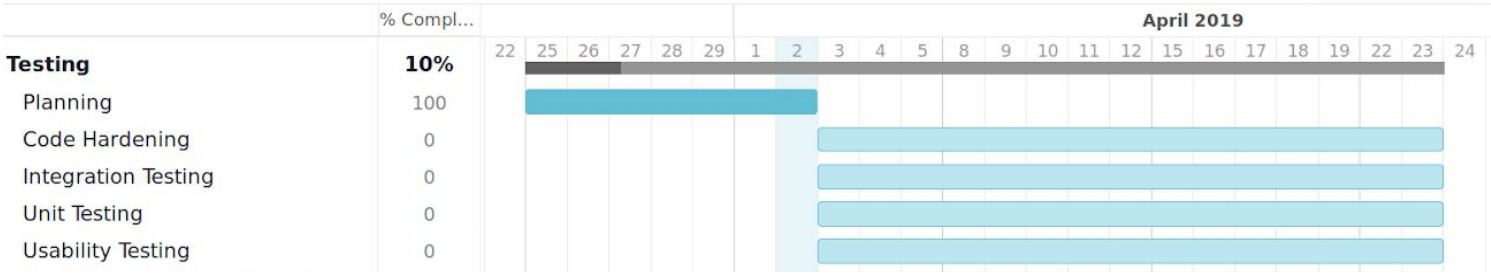

Figure 1.

In figure 1. The displayed timeline we intend to provide for testing. We will be conducting our usability testing during the following three weeks.

5. Conclusion

Complete verification of software involves testing through multiple layers each with a specific focus. Our unit testing will ensure that each component is free from issues and handle invalid data being submitted. With unit testing any problems found can be quickly identified since each component is isolated during testing. With all modules of the application passing unit testing, the interactions between them can be verified to ensure that no regressions have caused breakages. Since our project is a image processing pipeline, this includes the successful exchange of data from the UI to the Airflow scheduler and back. Finally a correctly implemented design can be improved using the feedback gained from potential users during usability testing.

This multi-level approach to testing ensures that all aspects of the application are thoroughly tested. Additionally, it provides a simple method for identifying where problems are located. Because of the repetition of the test plan after any changes,

issues will not have a chance to require major code overhauls to correct. While we feel this design is suited to the new pipeline, improvements to the design can be incorporated based on user testing feedback.

Airflow, Docker, and Flask are new and innovative technologies that our team is excited to be working with. The pipeline technical prototypes are performing as expected and demonstrating the excellent integration potential of these technologies. As we continue to develop our product, the devised testing plan will ensure that our product continues to be functional throughout. Our team is confident that this design will meet the needs of USGS and will help them better provide an important service to the public at large.