Testing Plan

Version 1

4/3/20

CartoCosmos

Scott Ames, Jacob Kaufman,

Kaitlyn Lee, Christopher Moore

Sponsor:

USGS Astrogeology Science Center

Mentor:

Isaac Shaffer

Contents

1. Introduction

An important part of space exploration is launching satellites into space to orbit planetary bodies, collect large amounts of data, and take images of these bodies. The data and images are sent back down to Earth and used by the planetary science community to create planetary maps. These maps play a crucial role in understanding more about these bodies and their surfaces.

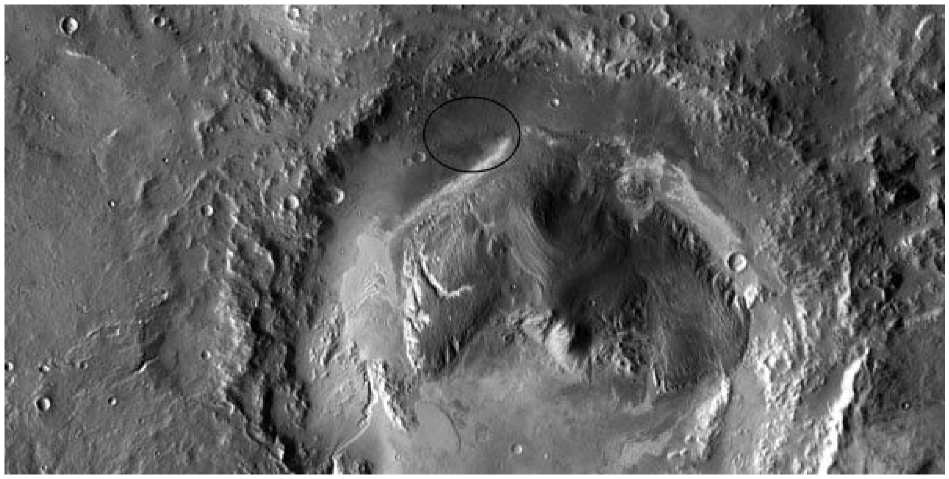
Figure 1: Mars Odyssey Map Highlighting Gale Center

Today, all eyes are set on Mars since it has been chosen as the next stop for manned space exploration. To accomplish this goal, one of the initial missions to Mars was the 2001 Mars Odyssey, commissioned to explore the red planet and transmit its findings back to Earth. It has orbited Mars since October 2001 and is still actively providing essential data today. The Odyssey's primary goals are to provide spatial data of the planet's surface, image surface minerals, and locate potential signs that the planet may have once supported life. See Figure 1 for a map of Mars created using the Odyssey data collected. Without this orbiter and the system of maps it has gifted us, we would have not been able to learn as much about Mars as we have.

The Mars Odyssey mission is just one example of all missions that are carried out. Because there are so many planetary bodies to explore, the planetary science community is very large and consists of scientists, students, and developers from different organizations spanning the globe. Some of these organizations include the United States Geological Survey (USGS), the National Aeronautics and Space Administration (NASA) and 10+ mission teams operating under NASA, other countries' space agencies like the European Space Agency (ESA) and the Japan Aerospace Exploration Agency (JAXA), and 30+ universities. All of these entities play an important role in the research process.

Our client is a small team from the USGS consisting of Trent Hare (Cartographer) and Scott Akins (IT Specialist). They both work for the Astrogeology Science Center (ASC), a sub-branch of the USGS, in Flagstaff, AZ. The ASC consists of developers, students, and scientists all working together to conduct research. Currently, there are approximately 20+ scientists, 20 mission-support/web developers, 20 IT specialists, and 10 students working for the ASC. The ASC is contracted by NASA to support the planetary science community, and they do so in different ways. The developers' main jobs are to create programs that aid in the map-making process and house the data collected during missions so that scientists do not have to store terabytes upon terabytes of data on their own machines. One job of the scientists is to create maps and work alongside planetary missions to conduct important research that would change the future of space exploration forever.

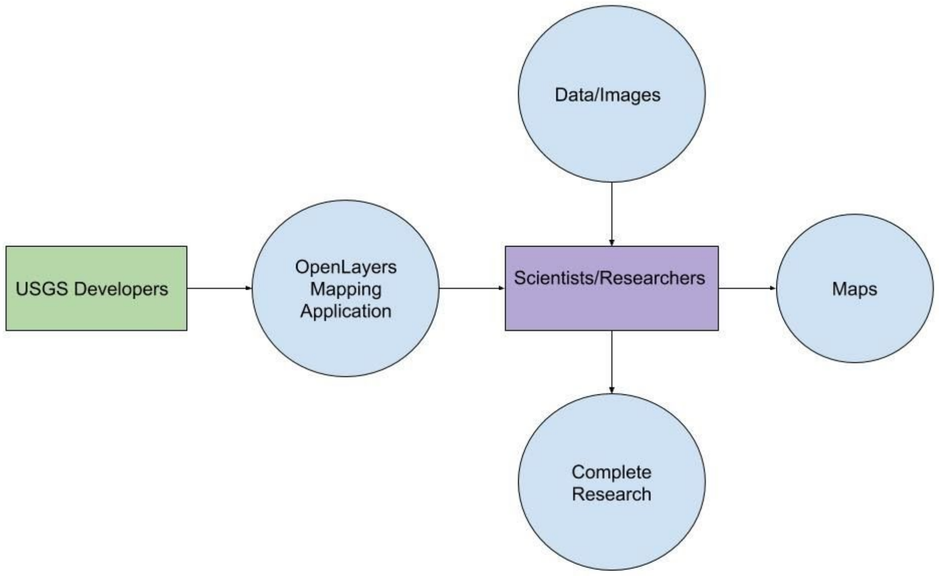

Figure 2: USGS Workflow

1.1 Problem Statement

The main workflow for the USGS scientists when creating maps and doing research is outlined in Figure 2. The scientists and researchers use the data collected during the missions to create maps of planetary bodies. The scientists then use a mapping application to view the maps virtually and are able to finish their research. Because the USGS scientists are mapping many different, complex planetary bodies, there are a few problems with these mapping applications:

- They only support maps of Earth.
- They do not allow users to change their latitude and longitude (lat/lon) settings
- They do not allow users to change what projections the current map is in.

1.2 Solution Overview

In order to solve the problems mentioned above, we are creating a JavaScript (JS) Node package that can be easily installed and used. This package will

- Contain a new interactive map built with Leaflet, an open-source mapping package, wrapped in a new modern graphical user interface (GUI).
- Contain a module that wraps the projection and lat/lon back-end code for use in other mapping applications.
- Allow users to view all bodies supported by the USGS.
- Contain lat/lon buttons to change the lat/lon settings.
- Allow users to change the projection of the map.

In addition to the Node package we will be creating, the USGS wants us to create a Python solution to be used in Jupyter Notebooks and a search bar with an auto-complete function to search for all named features on a target.

1.3 Testing Overview

Since we are creating virtual mapping applications catered to scientists, we must ensure that all the data is accurately displayed. In order to do so, we will be testing our project. Software testing is the process of writing and running tests on parts of a codebase to ensure the quality of the code. In general, it is very important that all codebases be tested vigorously to find bugs before they can cause any harm to users. In order to test our codebase, we will conduct a series of tests which fall under three categories:

- Unit Testing
- integration testing
- Usability Testing

Each category of testing will enable us to test our many modules in different ways. Unit testing will allow us to test each class's individual methods to ensure that each part of every module works as it's supposed to. Most of the classes we are unit testing deal with the data that must be correct in order to display the maps correctly. After all of our unit tests pass, we can begin with integration testing: testing that each of our classes and modules connect and transmit data to each other correctly. Since our modules are made from many different complex classes that all connect, we must test how all these components interact with each other. Finally, we will have a huge emphasis on usability testing: having users go through a user manual and record their experiences with the product. Usability testing is very important to both us and our clients since this product will be used by scientists all over the world. It is important to make something that users will be able to use and understand. In order to do our testing, we will be using scientists with different levels of technology experience and people not in the planetary science community. This way we can ensure that our product is easy-to-use and learn.. In the following sections, we will outline how we plan on executing the three testing categories beginning with unit testing.

2. Unit Testing

Unit testing is used to ensure that individual functions of a piece of software are tested and working properly. Most unit tests are very simple and only require an input and output verification. Other unit tests are used to test the creation of objects and verify that the object contains the correct information. Unit tests will allow any errors at a fundamental level to be found and corrected. Since our project has three separate modules (Leaflet, Jupyter Notebook, and Autocomplete) we have created three different testing plans for each module.

In order to test our Leaflet module, we will be using the testing frameworks Jest, React Testing Library, and Mocha. Jest and React Testing Library will be used to verify the creation of our GUI since it is written using React. Jest is a testing framework that allows you to access DOM elements. The React Testing Library is a set of helper functions that will allow us to test our React components easier. The main JavaScript code in the Leaflet module will be tested using Mocha. Mocha is a feature-rich JavaScript testing framework running on Node.js. Since our Jupyter Notebook module is a copy of our Leaflet implementation (without projections) but written in python we are going to be using PyUnit to test this module. PyUnit is an easy to use framework we can use for both unit testing and integration testing. Lastly, we will also be using Mocha in order to test our Autocomplete module.

2.1 Astro

The Astro module contains the entire back-end functionality of the planetary map. This is where we make calls to the USGS's GeoServer, create the Leaflet map, and add support for changing projections and lat/lon settings. It contains a mixture of classes inheriting from Leaflet classes and standard JS classes and follows an object-oriented paradigm. Every class will have its own Mocha unit testing code. This will ensure every class works in its intentional way.

2.1.1 AstroMap

The AstroMap class is the main class of the module and utilizes the other classes described below. It will inherit from L.Map and be used in the same way as it's parent. That is, AstroMap will need layers (L.Layer), a coordinate reference system (CRS) or projection, and controls (L.Control) to be added to it. In order to support non-Earth projections, we will be utilizing the package proj4leaflet. This package adds the class L.Proj.CRS to create non-Earth projections. Because it inherits from the Leaflet base map class, it inherits all of the variables and methods of L.Map. The Mocha unit tests for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Available Target for Map Creation	Test the map creation process for a supported target.	**“mars”, “MARS”, “Mars”**	An AstroMap object without errors.
Unavailable Target for Map Creation	Test the response to an unsupported target.	**Any string value.**	Error stating the target entered is not a supported target.
Change Current Map Projection	Test that the map properly changes its projection.	**“north-polar stereographic”, “cylindrical”**	The map’s CRS, center, and view are changed and set correctly
Unavailable Projection	Test response to an unsupported projection.	**Any string value.**	Error stating the projection entered is not supported.
Check the Available Projections for a Target	Test that the hasNorthPolar and hasSouthPolar methods return true.	**N/A**	Instantiate a map with Mars and methods return true.
Check the Available Projections for a Target	Test that the hasNorthPolar and hasSouthPolar methods return false.	**N/A**	Instantiate a map with Dione and methods return false.
Get Target Name	Test that the name of the target is being set and stored correctly.	**N/A**	Instantiate a map of Mars and target() should return Mars.
Get Current Projection	Test that the name of the projection is being set and stored correctly.	**N/A**	Instantiate a map of Mars and projection() should return cylindrical by default.
Set and Get Current Layer	Test that the set and get current layer method work round trip.	**WMS Layer**	Instantiate a WMS layer object, set it to the current layer, and check that it is returned in the getter method.
Incorrect Object to Set Layer to	Test response to setting the current layer to a non WMS layer object.	**Any Object.**	Error stating that the method only takes in WMS layer objects.

2.1.2 Layer Collection

This class stores the base layers (main map layers) and overlays (surface features) for a specific target and projection for quick access and ease of use. As stated previously, layers for a map are stored on the USGS's GeoServer. A JSON was given to us by the USGS that contains the parameters to query the Geoserver for all layers. These parameters include the URL and the display name of the layer. The Mocha unit tests for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Add to AstroMap	Tests the ability to add the control to the AstroMap Object.	**AstroMap Object**	An AstroMap object with Layer Control attached with no errors.
Available Target for Layer Creation	Test the layer creation process for a supported target.	**“mars”, “MARS”, “Mars”**	A dictionary holding all the correct layers for the current target and projection.
Unavailable Target for Layer Creation	Test the response for an unsupported target in the layer creation process.	**any string value.**	Error stating the target entered is not a supported target.
Available Projection for Layer Creation	Test the layer creation process for a supported projection	**“L.CRS.EPSG4326”**	A dictionary holding all the correct layers for the current target and projection.
Unavailable Projection for Layer Creation	Test the response for an unsupported projection in the layer creation process.	**Any string value.**	Error stating the target entered is not a supported projection.

2.1.3 Projection Control

The Projection Control class handles the back-end when a user clicks on the projection buttons. Since this class inherits L.Control, it is added to the AstroMap in the same way as other controls, like the zoom control. It maps proj-codes (the way scientists classify projections) to their names:

- Cylindrical -> `EPSG:4326`
- North-Polar -> `EPSG:32661`
- South-Polar -> `EPSG:32761`

The Mocha unit tests for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Add to AstroMap	Tests the ability to add the control to the AstroMap Object.	**AstroMap Object**	An AstroMap object with Projection Control attached with no errors.
Switch to North Polar Projection	Tests the ability to switch to a North Polar Projection	**N/A**	An AstroMap object without errors and the CRS set to `EPSG:32661`.
Switch to South Polar Projection	Tests the ability to switch to a South Polar Projection	**N/A**	An AstroMap object without errors and the CRS set to `EPSG:32761`.
Switch to Cylindrical Projection	Tests the ability to switch to a Cylindrical Projection.	**N/A**	An AstroMap object without errors and the CRS set to `EPSG:4326`.

2.1.4 Coordinate Control

The Coordinate Control class handles the back-end when a user clicks on the lat/lon buttons. Since this class inherits L.Control, it is added to the AstroMap in the same way as other controls, like the zoom control. The Mocha unit tests for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Add to AstroMap	Tests the ability to add the control to the AstroMap Object.	**AstroMap Object**	An AstroMap object with Coordinate Control attached with no errors.

2.1.5 AstroMath

The AstroMath class is a helper class that calculates different longitude and latitude domains and ranges for a specific target. It uses the JSON that was given to us by the USGS to grab the current target's radii and stores them in local class variables. Using the target's radii will allow conversion from planetocentric and planetographic latitudes. The Mocha unit tests for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Available Target for AstroMath Initialization	Test the AstroMath class initialization.	“mars”, “MARS”, “Mars”	Correct Major and Minor radius for the target.

Unavailable Target for AstroMath Initialization	Test the response for AstroMath initialization for an unsupported target.	Any string value	An error message stating the target is not supported.
From Degrees to Radians	Test the conversion method from degrees to radians	Any double value.	The converted value from degrees in radians.
From Radians to Degrees	Test the conversion method from degrees to radians	Any double value.	The converted value from radians in degrees.
Latitude Converted to planetocentric	Test the conversion method from planetographic to planetocentric Latitude	Any double value in a -90° to 90° range.	The converted value from planetographic to planetocentric latitude.
Latitude Converted to planetocentric	Test the error message for the conversion from planetographic to planetocentric Latitude	Any number value not in a -90° to 90° range or any value that is not a number.	An error message stating the number is out of the range or the input is not a number.
Longitude Domain Converted to Positive West	Test the conversion method from positive east to positive west longitude.	Any double value in a -180° to 180° range.	The converted value from positive east to positive west longitude.
Longitude Domain Converted to Positive West Over Boundary.	Test the error message for the conversion from positive east to positive west longitude.	Any number value not in a -180° to 180° range or any value that is not a number.	An error message stating the number is out of the range or the input is not a number.
Longitude Range converted to 0 to 360	Test the conversion method from -180° to 180° to 0 to 360° longitude range.	Any double value in a -180° to 180° range.	The converted value from -180° to 180° to 0° to 360° longitude.
Longitude Range converted to 0 to 360 Over Boundary.	Test the error message for the conversion from -180° to 180° to 0 to 360° longitude range.	Any number value not in a -180° to 180° range or any value that is not a number.	An error message stating the number is out of the range or the input is not a number.

Wrap Longitude Value	Test the ability to wrap a longitude value higher than the accepted range.	Any double value.	The converted longitude value in the range -180° to 180°.

2.1.6 AstroProj

AstroProj is a helper class that queries for proj-codes and proj-strings for a specific target and projection. Proj-codes are unique identifiers given to a projection. Proj-strings are used to define projections and their centers and units. Because the USGS's GeoServer only accepts Earth proj-codes in queries, we must use Earth proj-codes instead of a target's real proj-code. The Mocha unit tests for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Get Radius	Test the ability to get a target's minor and major radius values.	"Mars", "MARS", "mars"	A dictionary with the major radius on a and the minor radius on c.
Target not Supported	Test the response to an unsupported target.	Any string value.	Error stating the target entered to try to get its radii is not a supported target.
Get Projection String and Code	Test the ability to get the correct projection string for a target given the projection.	"Mars", "MARS", "Mars" AND "northPolar", "southPolar", "cylindrical"	Dictionary containing the correct proj-string and code for the target
Projection not Supported	Test the response to an unsupported projection.	Any string value.	Error stating the projection entered is not supported.

2.2 Jupyter Notebooks

The Jupyter Notebook module contains two classes used to create Leaflet maps, controls, and a graphical user interface. In order to achieve this functionality, we designed the Jupyter Notebook module using ipyleaflet, a Python Leaflet wrapper, and ipywidgets, python GUI elements.

2.2.1 Astro_Map

The Astro_Map class is the main class of this module and utilizes the Astro_GUI class for the front-end and back-end functionality of the different on-screen buttons and selectors. This class takes a specific target name and uses ipyleaflet to create and display a map of the target. The PyUnit tests for this module are described below:

Unit Test	Purpose	Sample Input	Expected Output
Available Target for Map Creation	Test the map creation process for a supported target.	“mars”, “MARS”, “Mars”	An astro_map object without errors.
Unavailable Target for Map Creation	Test the response to an unsupported target.	Any string value.	Error stating the target entered is not a supported target.
Find Radii for Target	Test that the correct a and c radii are returned.	“mars”	Return Mars’ radii.

2.2.2 Astro_GUI

The Astro_GUI class creates and monitors the latitude and longitude selectors for the planetary maps in Jupyter Notebook. It does this by utilizing the python module ipywidgets. The PyUnit tests for this module are described below:

Unit Test	Purpose	Sample Input	Expected Output
astro_gui Class Initialization	Test the initialization of the GUI class	N/A	astro_gui class object without errors.
Well-Known Text Button	Test the creation of the well-known text button object.	N/A	An ipywidget button object.
Well-Known Text Box	Test the creation of the well-known text box object.	N/A	An ipywidget text box object.
Longitude Direction Selector	Test the creation of the longitude direction selector object.	N/A	An ipywidget selector object.
Draw Label	Test the creation of the draw label object.	N/A	An ipywidget label object.
Coordinate Label	Test the creation of the coordinate label object.	N/A	An ipywidget label object.
Latitude Domain Selector	Test the creation of the longitude domain selector object.	N/A	An ipywidget selector object.

Longitude Range Selector	Test the creation of the longitude range selector object.	N/A	An ipywidget selector object.

2.3 AutoComplete

The AutoComplete is responsible for providing the full names of features on a target after they are partially entered inside of a search box on the USGS website. The planetary feature names will be pulled from the USGS's GeoServer, and when they are selected in the AutoComplete, we will bring up the map of the target with the requested feature. Users will be able to search for all features from the main USGS website page, and target-specific ones from a target's page. The Mocha unit tests are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
AutoComplete Information Query	Test validity of information returned by AutoComplete Query.	"Olympus Mons" "Olympus" "Oly"	Returns features with sample input in their name.
AutoComplete Results List	Test displaying results form information query.	N/A	Feature names returned from query are displayed as a list.
AutoComplete Selection Query	Test selection of result opening feature page.	"Olympus Mons" "Aida-Wedo Dorsa" "Po Chü-I"	Moves user to feature page on USGS website.

The creation of these unit tests will ensure every individual part of our product for USGS works as it should. In the next section, Integration Testing, we will discuss our testing plan so we can ensure each component works together as it is intended to.

3. Integration Testing

Integration testing is used to verify that all the components of a project are working together how they are intended to. By using integration testing, we can ensure that each module is delivering the correct data to the other modules in the project. This is going to be really important with our virtual planetary mapping application because it is composed of many different parts and controls. This level of testing will show us if one of our modules is failing to deliver the correct information to another module. For our JavaScript modules (Leaflet and AutoComplete Modules) we are planning on using Mocha.js to do our integration testing. Using Mocha.js we will be able to compare different instance variables and the data they are holding. For our Jupyter Notebook modules, we are planning on using the testing library pytest. For each module we will outline each integration test we need to perform, the components involved, and the expected output assuming the interactions take place correctly.

3.1 Leaflet App

The integration tests listed in this section will be for all the components in the Leaflet module, including the graphical user interface. Each subsection will describe a connection that needs to happen in order to correctly display our planetary maps.

3.1.1 Layers to AstroMap

In order to get the layers and overlays for the planetary targets, the AstroMap class must connect with the LayerCollection class. Testing this connection will ensure that the AstroMap gets all the possible layers and overlays for each specific target.

Sample Input/Action	**Expected Output**
Create a Layer Collection and add it to the map.	Layers and overlays added to the AstroMap

3.1.2 Projection Control to AstroMap

The projection control class tells the AstroMap what projection (cylindrical, north polar, and south polar) the map should currently be in. Testing this class will verify the AstroMap will always be in the correct projection.

Sample Input/Action	**Expected Output**
Set the map to each projection: “cylindrical”, “northPolar”, “southPolar”	Projection Control sets the correct map projection.

3.1.3 Projection Control to Layer Collection

When AstroMap switches to a north or south polar projection, the LayerCollection class must get the available layers and overlays for that target and projection. Testing this process ensures that the AstroMap gets all the possible layers and overlays for each projection and updates its layers for that projection.

Sample Input/Action	**Expected Output**
Change the map’s projection: “cylindrical”, “northPolar”, “southPolar”	A dictionary container the layers and overlays for that specific target and projection

3.1.4 Coordinate Control to AstroMap

The coordinate control class must connect to the AstroMap so that it can retrieve the user's current mouse position coordinates from the AstroMap object. Testing this information is essential so that we provide the user with correct mouse position coordinates.

Sample Input/Action	**Expected Output**
AstroMap mouse over object.	List with longitude and Latitude of current mouse position

3.1.5 AstroProj to AstroMap

The projection control class gives the AstroMap class the correct proj-string so that users can view supported planetary targets in both north and south polar projections. Testing this information will verify that our proj-strings create the correct maps.

Sample Input/Action	**Expected Output**
Proj-string: **“"+proj=longlat +a=" + radii["a"] + " +b=" + radii["c"] + " +no_defs"”**	The correct map of the current projection

3.1.6 AstroMath to Coordinate Control

AstroMath contains all the necessary conversions the coordinate control needs in order to present different planetary longitudes and latitudes. Testing the connection of these components will ensure that users can view mouse position coordinates in different ranges, domains, and directions.

Sample Input/Action	**Expected Output**
Coordinates: (123, 90)	The Coordinate Control class will ask the AstroMath to convert the longitude and latitude based on the user selections. The AstroMath class must report the correct conversions based on the users choices.

3.1.7 Projection Control to Projection Switcher

The Projection Control connects with the GUI element Projection Switcher to allow users to click on the projection switcher and switch the current projection of the map. Testing this interaction will verify that users can switch to different projections or a planetary target.

Sample Input/Action	**Expected Output**
onClick DOMEvents	The correct projection map based on the projection the user selected.

3.1.8 Coordinate Control to Latitude and Longitude Switcher

The GUI element Longitude and Latitude Switcher allows users to switch between different domains, directions, and ranges by connecting with the Coordinate Control. Testing this interaction will verify that users can switch to different longitude and latitude types.

Sample Input/Action	**Expected Output**
onChange DOMEvents	The correct mouse position coordinates based on the coordinate type the user selected..

3.1.9 AstroMap to Map Container

The most important connection in our project is displaying the map. The AstroMap must connect with the Map Container to have a planetary map show up on any website. Testing this connection will ensure that our maps will be visible on a webpage.

Sample Input/Action	**Expected Output**
AstroMap	AstroMap attached to the map container DOM element

3.2 Jupyter Module

The integration tests listed in this section will be for all the components in the Jupyter Notebook module, including the graphical user interface. Each subsection will describe a connection that needs to happen in order to correctly display our planetary maps.

3.2.1 Layers to Astro_Map

In order to get the layers and overlays for the planetary targets, the AstroMap class must connect with the layers. Testing this connection will ensure that the AstroMap gets all the possible layers and overlays for each specific target.

Sample Input/Action	**Expected Output**
A dictionary with two lists, one for the layers and another for the overlays.	Layers and overlays displaying on the Astro_Map

3.2.2 Coordinate Control to Latitude and Longitude Switcher

The Longitude and Latitude Switcher allows users to switch between different domains, directions, and ranges by connecting with the Coordinate Control. Testing this interaction will verify that users can switch to different longitude and latitude types.

Sample Input/Action	**Expected Output**
OnChange Widget Events	The correct mouse position coordinates based on the coordinate type the user selected.

3.3 AutoComplete Module

The integration tests listed in this section will be for all the components in the Autocomplete module. The AutoComplete will pull the target that is currently displayed on the map. Each subsection will describe a connection that needs to happen in order to correctly display the feature names for a target on the map.

3.3.1 Pulling Feature Names from Server with Target as a Parameter

To use the Autocomplete, we have to pull the correct feature names from a USGS server for the currently viewed target. Testing the connection to get the data and the validity of the data will be used to verify we are getting what the user needs.

Sample Input/Action	**Expected Output**
Current Target	List of target feature names.

3.3.3 Sending User to Feature Page On Selection

When a feature name on the AutoComplete list is selected by the user, they will be taken to that feature's information page on the USGS website. Testing this connection will show the autocomplete is useful and will actually save the user some time in getting to a feature.

Sample Input/Action	**Expected Output**
OnSelection Results List Event	Page is moved to the feature page for the selected feature name.

4. Usability Testing

Usability testing concentrates on the user's experience with a software system. The end goal is to ensure that our mapping applications are both comfortable and meet the functional needs of the target user base. This category of testing will highlight the logical bugs and user errors that our mapping applications must be able to handle. We will be testing both our Node package and our Jupyter Notebook Module.

Looking at our user group, we have two end-users for our system: scientists and general users. Scientists are going to be users looking to use our planetary mapping application to conduct research based on the maps we are displaying. We will be using 6 scientists with different technological knowledge. General users, on the other hand, could be anyone just trying to look at maps of planetary bodies virtually. We will be using 4 general users. Understanding that we have two end-users, we are going to conduct tests using each of these groups to gain insights on how we can fix our system to meet both our user's needs.

In order to conduct our test, we will be conducting and recording a 30-minute interview with each end-user group where the user will install and use our mapping applications. Some users will download our Node package and other users will download our Jupyter Notebook module. Even though the testing process is very similar for both modules, we are going to keep the testing separated. Using this method, we can gauge the user's experience for both modules. We are going to have the user use the main components of our system and express their thoughts about the overall aesthetics and functionality. In the next subsections, we will outline how each user will install and use our mapping application.

4.1 Installation

During the installation process, users will install our packages so that they are able to use them. The users will follow our user manual, which will give them each step needed to install our packages. Each user will be asked to express their confusion and comforts with each step of the installation process. This will allow us to gain insight into which parts of the process are confusing. The installation tasks are described below:

- Leaflet Node Package Test (Assuming NPM is installed)
  - The user should access our Node package installation user guide
  - The user should install the CartoCosmos Node package
- Jupyter Notebook Module Test (Assuming Jupyter Notebook, PIP, and Conda are installed)
  - The user should access our Jupyter Notebook installation user guide
  - The user should download our environment.yml from our GitHub

    - The user should PIP install CatroCosmos

4.2 Viewing Maps

During the viewing process, users should view maps of different planetary bodies and use the components of our application to change the viewing process. The user can switch from different targets, change between different lat/lon types, switch projections, and swap overlays and layers of each target. All these tasks are laid out below:

- Leaflet Node Package Test (Assuming NPM is installed)
    - The user should access our Node package user guide
    - The user should pick and view a target that is supported by the USGS
    - The user should switch projections if they are available for that target
    - The user should hover their mouse over the map and see coordinates
    - The user should see and use the scale bar
    - The user should switch layers using the layer switcher
    - The user should toggle on and off overlays using the layer switcher
    - The user should switch longitude range to 0 to 360
    - The user should turn the longitude direction to positive west
    - The user should change the latitude domain to planetocentric
    - The user should go into fullscreen
    - The user should change between targets
    - The user should comment on the aesthetics of the GUI
- Jupyter Notebook Module Test (Assuming Jupyter Notebook, PIP, and Conda are Installed)
    - The user should be able to access our Jupyter Notebook user guide
    - The user should pick and view a target that is supported by USGS
    - The user should hover their mouse over the map and see coordinates
    - The user should see and use the scale bar
    - The user should switch layers using the layer switcher
    - The user should toggle on and off overlays using the layer switcher
    - The user should switch longitude range to 0 to 360
    - The user should turn the longitude direction to positive west
    - The user should change the latitude domain to planetocentric
    - The user should go into fullscreen
    - The user should change between targets
    - The user should comment on the aesthetics of the GUI

4.3 Expected Outcome

For the user testing, the main expected outcome is that users are able to view and interact with a map without any problems arising. We also expect some comments from each user that includes what they liked/disliked and if there are any features missing that they would like to see implemented into the GUI/map. We hope to use these comments to improve our application from a user's perspective. Users should also be able to follow along with our User Manual without any problems. From the users' comments, we will be able to analyze our user manual and improve it if needed.

5. Conclusion

The planetary science community, which is very large and spans the globe, consists of both developers and scientists that work together to create maps. Without these maps, planetary missions would be impossible. Our clients, Trent Hare and Scott Akins, who work for USGS, are scientists and developers who help in the map-making process. In order to do research with these maps, USGS scientists need a mapping tool that allows them to view non-Earth bodies, change the projection of the map, and change the lat/lon settings.

In order to solve these problems, we are creating a new mapping tool with Leaflet and a portable package. The new Leaflet tool will contain extra functionality such as the lat/lon switcher and projection switcher. The package will contain the back-end code for the projection and latitude and longitude switches.

Currently, we are done with the prototype and have implemented all requirements of the project. We are now starting the testing phase of the project. By using unit testing, integration testing, and usability testing we can ensure our mapping tool and portable package are functioning properly. These series of tests will also verify that our packages are ready to deliver to the USGS at the end of the semester. The feedback of the three phases of testing will lead to a very successful implementation of our mapping tool and portable package.