**Satellite Shade Analysis**

**Test Plan**

**Prepared for: Dr. Eck Doerry**

**Prepared by: JBOP Solutions**

Taylor Born,

Ben Johnson,

Steffen Olson,

Rachel Pang

Version 1.10

Last Updated: April 30th, 2014

# **Table of Contents:**

Introduction 1-3

Unit Testing 3-5

Integration Testing 5-8

Usability Testing 8-9

Conclusion 10

# **Introduction**

Electrical power has become an ever more important resource in our modern, information-based economy, with average household consumption rising steadily to power an ever-increasing array of high tech devices. Up until recently, people have relied solely on power plants powered by fossil or nuclear fuels to provide power to their homes, which contribute to pollution, waste and ecological impacts. Solar power has been a growing trend in providing clean, reliable energy for homes and businesses, and since its introduction in the 1970’s, it’s now more affordable and more accessible for everyday people to invest in. Because of this, interest in solar installation has skyrocketed.

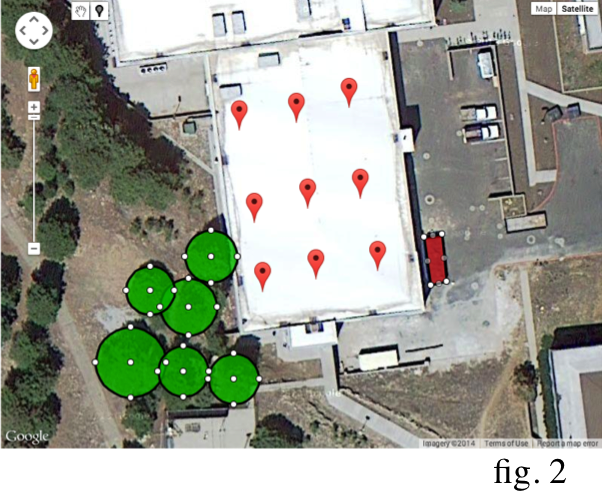
Rooftop Solar is one of these solar paneling companies that is located in Flagstaff, Arizona that deals with the sales, marketing and installation of solar panels on buildings. On average, they receive around 30-40 clients a month that may end up being potential adopters of solar energy, and give them the information and estimations needed to make a decision on whether it is right for them. They are just one of hundreds of the companies throughout the United States involved with the effort to make energy production greener.

## The Current Process of Shade Estimation

There’s a lot of information to take into account when estimating the solar energy one would receive from paneling. There’s basic knowledge required such as the angle of the roof, as well as the amount of solar paneling desired. However, the hardest part of the estimate relies on obstructions around the building, such as trees and other buildings. 

To estimate this shade, a device called a SunEye (see fig. 1) is used to take an upward fish-eye lens photo of where a panel is planned to be installed, and is done where every panel would be installed to capture any surrounding obstructions. This data is then run through an algorithm that uses sun positioning throughout the course of a year to estimate how much sunlight hits those panels (in a percentage) over the entire year. The amount of shade that hits the panels can make or break an investment of solar energy, with a bad estimate possibly costing an adopter more money than they would recoup from the solar paneling.

## Our Alternative to Performing Shade Estimations

Our selected capstone project deals specifically with the inefficiencies of the current methods solar companies around the world use in order to get an initial shade analysis estimate of a client’s roof. The current process involves sending employees on the roof of clients’ buildings to do the analysis with the SunEye. This is not only a huge use of time, as it requires employees to go on site to every potential contract, but a safety hazard for employees, as they need to climb up on every roof and perform the solar eye analysis.

Our project allows the company to perform these analyses through a web application. This application will take in an image from Google Maps, and allow the user to mark where obstructions are seen in the image, along with where the solar panels plan to be placed. The application will then use this information, with their relative heights and the image scale, and calculate how much percentage of shade will cover the panels over the course of a year (see fig. 2 for an screenshot of our front-end). Right now, our solution provides results very close to the SunEye and can serve as a very viable alternative to performing initial estimates.

Besides the calculation application, our project also encompasses a variety of extra features which includes a secure user system, which allows for logging in with a Google Account, and a project creation system that includes information about the locations (client name, address, extra information) and the locations of each of the user marked obstructions and solar panels in the images.

To keep track of all this information accurately, we are using databases implemented through Google App Engine; this will store information related to the current user logged in, as well as the information stored on each of the projects created. This project is contained on a Google appspot website that we will create and will eventually be directed to on Rooftop Solar’s website for them to license out to other companies later on.

## Our Created Alternative's Goal

This application will give an accurate estimate on how much shade will hit the panels throughout a year, solar companies will be able to give this information to potential clients/buyers to make an educated decision on if solar panel installation would be appropriate for their home. This would lead to a contract signing for the purchase and installation of solar panels, or save time in not having to send an employee on site to perform a shade estimate.

With future plans from Rooftop Solar to monetize and license out the application to other companies in the future, this application has the potential to save the time of solar companies around the globe in performing an accurate shade estimate, while also saving money and preventing accidental employee injury from climbing up on more rooftops.

## An Overview of our Test Plan

Our plan for testing primarily focuses on testing any major feature that we have data-sensitive, while ensuring that all the major parts work seamlessly together. Unit testing will involve most of the python files created for this project, relating to the modification and creation of user projects, as well as the algorithm in order to calculate the shade percentage and ensuring their functionality works as intended in various different test cases.

Once unit testing has been done, we plan to perform integration testing between all major components of our project. This will encompass communication between the user projects, the shade calculator, and the front-end website. Most of this functionality involves the correct sending and receiving of data between components, and whether the functions get called at the right time.

Finally, the last part of our test plan focuses on usability testing. This will be done to ensure that all the functionality of our project is explicit and can be done by a solar industry worker who is new to the application, which may require minor changes to the front-end layout, as well as more instructions. We also plan on letting the workers at Rooftop Solar user test the application, and see if there’s any changes they would suggest to be made to make it better and more user friendly. All these sections on testing will be given in full detail in the following document.

# **Unit Testing**

In regards to the unit testing, the most important part about our project is the implementation of our algorithm and making sure it works correctly. If the algorithm does not work as intended, or is not accurate, then our project is useless. Because the purpose of our project is to replace the use of the SunEye for initial estimates, testing needs to be done on each part of the algorithm to ensure that the algorithm works without any unintended changes.

In addition, as a lot of the python files relating to project modification and loading pass data to each other fairly consistently, most of these files will also need to be tested in order to ensure that the data they’re working with does not get changed unintentionally. The following sections will explain how we plan to test each of these parts.

## Algorithm Unit Testing

To begin, the algorithm is passed data for the project in question. The latitude that it is given is used to generate sun angles for the course of a year in some time intervals, for the location on the Earth. The generation of these sun angles can be isolated and tested against known data of latitudes and corresponding positions of the sun during specific times in the year. The next step that the algorithm does to prepare for sun access calculation is to convert the data (including the obstructions and modules) it is given, to a context that it can work with. This mainly deals with positions of things. Positions are given to the algorithm as a distance and bearing from some constant point. The relative placement of items from the Google Maps interface needs to match the relative placement of the same items now in a new context.

The sun access calculation of the algorithm involves calculating intersections of lines. The intersect result needs to be accurate for all kinds of lines including vertical, horizontal, and parallel. Creating specific scenarios (placement of obstructions and modules) where these kinds of lines and pair of lines show up, can be tested for capturing the behaviour of the algorithm in handling these cases. For calculating sun access with trees, intersections are used to create mappings onto images of trees where it is determined if the spot of intersection hit the tree or was between tree branches etc. Testing intersection parameters and different tree attributes, the mapping on a tree image can be analyzed to ensure the spot it determined is in the correct vicinity.

## Project File Unit Testing

Project saving and loading is another critical part of our project. We will need to test all of the different components to ensure data consistency. There are three major components that need to be tested here.

First, the JavaScript front end that compiles all of the project data into a JSON formatted string and sends it to the server. We need to write tests that ensure the data being sent makes sense for the project markup. For example, if a user inputs 2 trees, one additional obstruction, and a solar module, we need to ensure that the output reflects that accurately.

Secondly, we need to test the part of the backend responsible for storing and loading the project data in the database. First we need to check that the data we received is in fact the same data that we expected to send. Then we need to make sure that the data we load from the database matches the data that we put in.

Finally, we need to test the part of the front-end that loads the data back onto the map. Again we first need to make sure the data we received from the server is the data that we expected. Then we have to check that the data resulted in the correct markup on the map. We’ll have to test this visually to make sure it looks exactly like it did when we saved it.

# **Integration Testing**

Ensuring the integration between all of our components in this project is key to ensuring that this project is reliable and works as intended. Our project consists of a few different large components working together without any trouble. These larger components include the user projects, the shade calculator, the user google account, and the front end website. To ensure a clean, issue free user experience, these components need to work together perfectly, by sending the correct information to the right component, where it will be read at the right time. This will be expanded upon in more detail in the following sections.

## Integration Between the User Google Account and the User Projects

Although this isn’t a large concern, as this functionality is primarily handled by the Google App Engine, there needs to be checks in place that ensure the correct user has access to their own projects, and no others. Currently, the implementation uses a similar implementation to a hash map, where each entity is distinguished by a unique key. The primary entity in the map is the user, which has a parent key associated with it. Each project entity created has its own individual key, but would contain the parent key information, and is essentially nested “under” the user entity.

In order to test this, our primary concern would be the implementation of how the user key gets created. The current implementation uses the user service provided by the Google App Engine, and uses a nickname function on the user object to serve in the creation of the key. Since the service uses Google Accounts, which are required for using this application, there shouldn’t be any conflicts with the nicknames in creating the same key, as the nickname is part of the “@gmail.com” e-mail address. However, this is something we will research and ensure that no two users will have the same nickname, making modifications as needed.

## Integration Between the User Projects and the Shade Calculator

Integration between the user projects and the shade calculator is a key concern for this application. The information inputted from the user into each project will need to be loaded into the calculator correctly, such as the address of the location or inputted obstructions, into the Google Map interface. Furthermore, on the shade calculator side, the method of saving and loading locations and sizes of obstructions in the database need to stay accurate to the inputted location and will not change to another area, which could potentially skew the results given when calculating the shade percentage.

Throughout our implementation of both the project system as well as the shade calculator, console logging was used at runtime to check certain inputs and outputs that both the projects and the shade calculator gave and received. As both components are completed, more formal testing will need to be done to ensure that everything works as intended. This section involves two separate parts: loading the project information and saving the project information.

Saving the project information involves the edit-project javascript and web page, as it takes the current location, the obstructions and module’s heights, as well as their size and location, and compiles them into a string. The information is then sent to the project, and saved. In order to test this part, we need to ensure that the data being sent to the project contains the correct coordinates and heights, and will not be changed when being converted to a string. This will involve creating some test cases for data that we specifically know, checking to see if saving and then loading the data changes any of the values.

Loading the project information into the Google Map for the calculator is done through the project listing web page when the user clicks the name of the project. This then uses the projectData python file to get the project data, load in the address’ coordinates into the Google Map on the edit-project web page, and then populate the image with the obstructions and modules. Testing this will be similar to saving, as we will take a case where we know the locations of certain modules and obstructions, and check to see if the obstructions are loaded in the correct positions, with the correct information.

## Integration Between the Front-End Website with the User Projects and Account

The front-end website is the primary means of which the user interacts with the application and its components. Therefore, in regards to the user profile, the interactions between the html pages, the Google App Engine functionality, and our own python files needs to be seamless without any unexpected errors.

In regards to the user account, the status of whether the user is logged in or not is currently shown on the layout bar. When the login button is clicked, or any page located in the “private” section, the user will redirect to the Google Account login, which is handled by the Google App Engine. Currently, the login implementation has not given us any problems, and likely will require minimal testing, as the implementation is based on Google’s API. One small thing we can ensure is that that the userStatus python file always shows the correct status of the user on the layout bar.

However, with the user projects, the entirety of the implementation was done through interaction between a variety of user project related python files, with the related web pages. The project listing web page, displays all the user’s specific projects, as well as a few text boxes for the creation of a new project. Although the text boxes already have code to check for incorrect project creation input, this likely can be expanded on to be more in-depth on input checking, as well as checking to see how many projects can be created by one user.

The project listing is handled by using projectListing javascript and python files. This essentially loads in every project associated with the user’s key, and displays it in a list that the user can read and analyze easily, and if there’s no projects, displays that there are no projects. We will be planning to test that all the information for each project is read correctly, and that there are no missing projects - projects that use the user key, but do not show up on the listing.

In addition to the information, each project in the listing have two different links: a project link, and a deletion link. The project link was explained earlier in the last integration section between the projects and the shade calculator, and the deletion link is a lot simpler. It uses a deleteProject python file to perform a deletion of the project from the user listing. The implementation currently asks for user confirmation if the user wants to delete the project, so testing will primarily revolve around checking to make sure the request made by the page removes all traces of the old project from the user’s listing.

## Integration Between the Front-End Website with the Shade Calculator

The front end website integration with the shade calculator revolves primarily around ensuring the data loaded onto the Google Map image gets sent to the calculator correctly. The unit testing for the calculate python file should handle most problems with the algorithm itself, however, the data sent is strictly coming from the stringifying of position and heights of each obstruction and module, and sending them to this algorithm using a function in the edit-project.js.

During the algorithm’s development, we implemented console logging to test different parts to ensure each part was being converted into the right format, however, we plan on doing more formal testing with various different locations as test cases to ensure that each obstructions’ size and location gets transmitted accurately to the algorithm.

Lastly, the edit-project web site also needs to receive and display the results given from the algorithm accurately in the results section. Testing this will be fairly simple, as we test various sites, we will compare the results given on the web page to a console output from the calculate python file.

Now that the unit testing, as well as the integration between all of the major components of this project has been discussed, this leaves usability testing as the last major form of testing we plan to accomplish before delivering our implementation.

# **Usability Testing**

Usability testing is an integral part of our project since the point of our project is to provide a better user experience for Rooftop Solar than their current method of obtaining their shade analysis. The main components of our usability testing will include creating and managing a project, and using the Google API markup to provide accurate calculations of how much shade an area would get. Users must be able to navigate through the website without external help. The website must provide all instructions necessary to perform the shade analysis. Our intended users are the employees at Rooftop Solar, so we must tailor the user experience to them-- therefore, our risks and project guidelines are outlined on the basis that only solar company employees will be our users and thus have a basic knowledge of what this tool is and how to gather the appropriate information.

## Creating, Editing, and Deleting Projects

Creating, editing, and deleting projects should be able to be performed by anyone. Throughout all three stages of the project, the process should be intuitive or provide enough instruction to allow the user to perform any action he or she desires. Each step should account for false, inaccurate data, or mistakes made by the user. We intend to have up to five pairs of people to navigate through creating a project, editing a project, and deleting a project. Five pairs of people is a reasonable number since we do not need a large pool of people to test, we just need to make sure there is a general consensus that they can perform these three tasks with the context clues they can find on the website. We would instruct the pairs to create a project with the given information, edit the project to change the information, and finally delete the project. We do not intend to provide any information on how to navigate through the website. We will watch how users interact with the product and make sure that the system is intuitive enough that we do not need to provide outside information to the user. This is to help us fix things like inserting instructions where needed and making sure the wording is clear and concise for someone to understand how to navigate through the interface. Once we are satisfied that our system works well, we will take it to our sponsor and have him and some of his employees test the system to make sure that our intended users can use the interface properly without outside instruction.

## Using the Google Map Markup Tools

The main feature of our application is having the user use the Google Map markup tools to outline trees, buildings, obstructions, and solar modules to outline where everything is on the map to perform the shade percentage calculation. It is important that the user knows how to use these tools properly since wrong inputs will result in inaccurate calculations and may lose Rooftop Solar potential customers.

There are not very many buttons on the Google Map viewer, but it is important that we provide information to the user on what these buttons mean so that they feel comfortable using the tool. We can use the same groups from the previous tests to mark all the obstructions given the heights of the nearby trees and obstructions. The only thing we would give them are the heights of the trees and see if they can navigate through the website and properly marked all the appropriate obstructions with the instructions given on the website.

Once we feel satisfied that our testing group has navigated through the site well enough, we would then ask our sponsor for data from sites that they have already visited and compare their results with the results that our project comes up with. The hardest part is relaying which tree or obstructions to mark as what height. For simplicity sake, we will just have users assume all trees and obstructions are X height. Our main concern is that they are drawing the trees appropriately

## Navigation and Account Login

There are not that many sections to our website, so it is crucial that users can deduce how to navigate through the website without a lot of help. This includes logging in to the site with their Google account information as well as navigating through the website to find out how to create a project. It is important that we make navigation easy so that Rooftop Solar employees are not confused as to how to use this website. This website is intended to be a solution to how difficult it is to gather shade percentage, not add to the problem.

We will have the same users that performed the above tasks report on if the feel and the flow of the website seemed appropriate. We tried to place all the main navigation buttons in an obvious place so people will have everything they need to get around the page in the navigation bar. It is important that the system does not feel clunky and impede the user from performing his or her task. If users have any questions throughout the site, it should be considered as a FAQ question and answer for the “Help” section of our website if it is not something we can immediately address such as a design issue or a misunderstanding of instruction.

# **Conclusion**

Our solution to the inefficiency in the Solar Industry requires our project to be as accurate to the SunEye as possible. If we want to ensure that this application will be a suitable replacement for solar companies to perform initial solar estimates, extensive testing of each component needs to be performed to ensure the incorrect information will not be given.

First, Unit testing will focus on the individual, key components of our project, relating to the user project system as well as the algorithm for the shade calculation. Next, integration Testing will require more involved test cases that ensure information from the larger components are sent and received accurately. Lastly, usability testing will focus more on the user experience and ensuring the application can be easily used and understood by anyone new to the system that is an employee of the solar industry

Although there has been some basic testing during implementation, in order to improve reliability and dependability of the application, we will perform formal testing that will provide some detailed evidence that everything in the project works as intended.