Requirements Specification

Version: 2.0

Team: Orion
Sponsor: USGS
12/10/18

Jun Rao

Brandon Kindrick
Chadd Frasier
Yuxuan Zhu

Table of Contents

Section 1: Introduction

Planetary Science is laying the foundation for humanity to become a space-faring species by discovering what our neighboring planets look like. In recent decades, NASA has achieved more development in the field of planetary science than anyone could have expected. What we can't ignore is the fact that the USGS has also played an important role in this. NASA worked closely with the USGS to meet all of their data processing and research needs. The USGS has spent more than 30 years developing the ISIS software to decode and examine these massive binary files from NASA. The precise topographic map produced by ISIS is used every time astronauts look for new potentially landing site on the Moon, Mars and asteroids. The USGSs' contribution to space exploration is its accurate foundational geospatial maps, and without them you could say astronauts are literally lost in space.

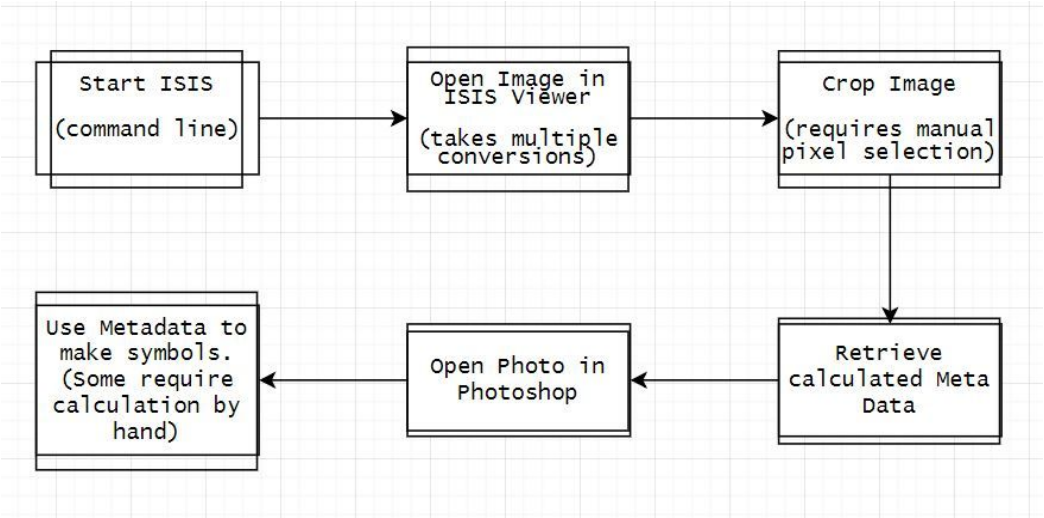
We are Orion and our team is proud to contribute to the Planetary Image Caption Writer project from the United States Geological Survey. Below I will introduce the members of our team and their respective responsibilities: Team members include Brandon Kindrick as team leader, Yuxuan Zhu as code designer and Chadd Frasier as version controller. Our client is Dr. Laszlo Kestay, he is a research geologist at the USGS, and his main job is to take data files from NASA and prepare them for publication using the ISIS3 software. His work can benefit the advancements of space exploration by NASA and accelerate the speed at which knowledge spreads.

After introducing our team and our client, I want to focus on the current workflow of the USGS. It is well known that images in publications are at the heart of sharing information. This is especially true for space publications because the images are key to seeing the geological features of the solar system. Most of the images of planetary surfaces in NASA’s archives have been processed through the USGS’s Integrated Software for Imagers and Spectrometers version 3 (ISIS3). The image tags of the ISIS3 file are filled with various basic geospatial information about the image such as latitude and longitude, the height of the Sun, the size of the pixels in relation to the planet, and so on. Unfortunately, there is no convenient way to extract this meta information so far other than by command line and a pencil. Scientists had to manually extract the metadata by using the binary editor on the ISIS3 file and then type into a document. The annotations on the image were manually built with third-party software like Photoshop after exporting data from ISIS3. There has also been backlash at NASA recently about the use of photoshop for research and it would be much better if they could say that the images came directly from the data processing software. This cumbersome process now leads scientists to basically lose access to the rich metadata contained in the ISIS3

image header. That's the topic we want to explore in detail.

Section 2: Problem Statement

In this section, I will start by sketching out our sponsor's workflow.



Knowing this workflow is really crucial to our understand of the sponsor's business. At first, our client needs to start ISIS3 by commands line, then he needs to select a MOC file which needs to be converted into an image projection, and that is how our client will convert binary image file to ISIS block file. This corresponds to the second step in the flowchart and following that he will crop the image, but in this process, he must preserve the pixel position in the image and then he inputs the pixel range into a function to crop the block. He cannot just select a location easily that looks good enough to publish. In addition, our client wants to use calculated metadata and set icons on the image, in order to do this, he needs to open a photo in Photoshop and finally use the metadata to create symbols like a scale bar and the Sun's position. Some of the icons need to be placed after human calculations. Through this description, we can find the problem is that our client is not happy about this timely process. This process could take anywhere up to 2 hours for a single publication image, making great figures very difficult to add to research papers quickly. Our client has used this software for a long time and he is ready for a change to his process. He found himself often not wanting to add images to his publications if they are not vital because of the cumbersome process. He wants an easier way to crop the image and add important icons such as a scale bar or north arrows to the image easily. Sadly this cumbersome process is currently slowing down the speed of publication and the speed of spreading scientific knowledge. As we

know, USGS has done a great job with the Integrated System for Imagers and Spectrometers(ISIS), but it is still slow and we can fix it. Our work on this project will be very important and needed if we want to increase the speed of exploration and discovery. In addition, ISIS3 can only be installed on Linux and MacOS machines at the moment, this is unfriendly to Windows users and limits the user base. So, our client, Dr. Kestay, also very much hopes to provide ISIS3 software for Windows machines.

Above are the two issues our client need to address; slow operation and limitations of the Windows platform. To create a quick and easy tool for all platforms, we will be designing a web application that will allow the server to do the grunt work of running ISIS3 while the user can have the speed of a web app.

Section 3: Solution Vision

In this section, we can turn to the solution. Before we come into the solution for this project, I want to first outline the issues we need to solve right now. Currently, many USGS scientists including our client think the process for editing images and metadata is too complicated and time-wasting, as well as the ISIS software does not support Windows operating systems. To solve these problems, we think a Web Application was the best way to go!

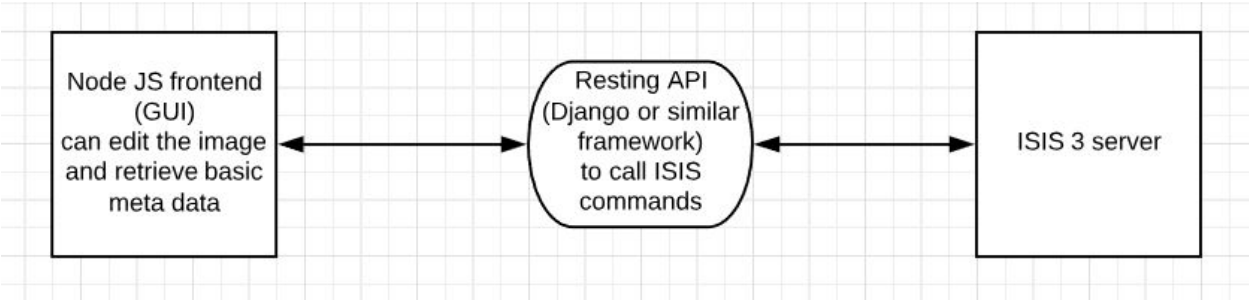
As we know, a web app must have its frontend and backend, so I will talk about our solution in detail from frontend and backend these two sides. The front end is very important to design well because our client clearly expressed the hope that having a user-friendly "viewer tool" for interactively exploring ISIS images and converting sub-images into publication-ready data and figures. We think the best way to go is to create a simple GUI. The GUI will allow users to view and interact with the data, as well as export images and metadata in an easy-to-use, standardized format. This will help both technical and non-technical audiences to operate easily without affecting the user base because of the technical skill gap. Our application will use Node.JS for our front end, because of the extensive libraries and performance speed. It also has libraries specifically for communication between a the front end web app and C++, which needs to be on the backend server.

In terms of the backend, we have known that ISIS uses C++ to decode massive binary files. The ISIS3 software has built-in functions for processing metadata, and editing images. ISIS3 is crucial for us to implement our app without “remaking the wheel.” As a result, our web app will use ISIS to calculate all the necessary data in the backend. We plan to run ISIS on a server for the backend. A server is required to run both ISIS and GUI. The server also relays messages between the server and ISIS. This is how the user can dynamically run the ISIS command and display the results of the command to the user.

Furthermore, in order to control all communications from ISIS to external applications. We chose to build a resting API using the Django framework because a resting API will do everything we need it to without having to edit the ISIS source code. Designing a resting API will save us time and networking heart ache. This API will let us call commands to the server using system commands which will make implementing the ISIS server much easier.

Secondly, we can not neglect the communication between our front-end GUI and the back-end server. In order to call the commands that are housed in the backend, we need to consider using HTTPS to create a Resting API that sits on top of the ISIS

server. To make it more clear, I will use a flow diagram to display.



In conclusion, our products need to be able to allow users to upload images, but the images that will be uploaded must be Map projected .cub images. This image is then able to be cropped, have metadata extracted and displayed through our GUI. In the uploading process, the image will be saved on the server. Our server will make a call to ISIS and it will process the image. After an image is processed by the back-end, it will send the calculated file back to the server, for the front-end GUI to then display the data. The server will also send relevant metadata to the GUI. Our front-end GUI will also have some interactive buttons that will use Metadata and display it under the image. Finally, our web app will allow users to export images and data to text files. We hope that this whole process will create a fast a secure data processing tool for scientists in the field to utilize.

Section 4: Functional Requirements

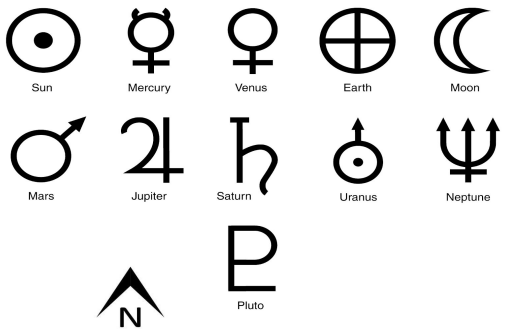
Before we discuss how the users will be expected to interact with the system and why we want it that way, we need to discuss what exactly the system needs to do to be classified as functional. We first must run the ISIS software on a Linux based server. This is because the current software is already tailored to UNIX and therefore can be expanded to a web interface much faster than editing source code to make it portable to Windows. Our product will allow users to uploading image files to the web app and receiving a .cub of that image back from the server. This is the most basic and most important interaction that our web application needs to be considered a success. If we cannot access the server and retrieve the image we need then the app will not be capable of creating publication ready images. The web app then needs to be capable of navigating the image to find desirable portions of the picture and then cropping it out. This will be handled by the frontend to ensure that the user will not have to wait for buffering on the image. Our product will need to have buttons or switches on the GUI that signify specific metadata parts to be included in the publication image. This is a User Experience requirement that will also need to be supported by some API calls. Lastly, our product will need to be able to export the image and metadata in format that can be pasted into publications. Images can be in any format that supports images in text such as JPG, PNG, or PDF. The metadata must be exported as a text file because that was in the project specifications.

In order to achieve these high-level requirements we will also need to complete some smaller more specific tasks. For example in order to make ISIS commands work from a message queue we will need to write an API for the ISIS software. An API will allow us to call any command in ISIS using a message from a server on the fly and the only thing we would need is a single Resting API create with Django or another similar framework. We can write a single API to control all the communication from ISIS to the outside app. The API will be capable of decoding the message from the queue and run the correlated commands in ISIS. The Resting API will then send the resulting data back to the frontend app to be displayed to the user.

In order for the API to receive messages from the app we will be implementing some type of queue system to receive the commands in the correct order they are sent. This can be done using HTTPS in a creative way. Since commands have to be executed in a specific order we can use Acknowledgement numbers to keep track of which message came first. This will also help us find errors in our data transfer. If an acknowledgement comes in out of order we can resend the packages that were lost. If we have trouble transferring large image files a possible solution is that we can always break images up into smaller packets to make the transfer easier to handle.

In order for our product to navigate the images we will need to implement and dynamic viewing window that is capable of showing multiple resolutions clearly without blurs or stretching. This can be done either by examining the metadata and calculating the pixel density of the range that is currently in the window using ISIS, which would be very slow or we could give the large image to the web app viewer and then just zoom in and out using Node.JS to adjust for clarity. After some research we determined that since our app needs to be fast and convenient to use we will want to implement a viewer that is similar to the USGS Pilot software map viewer. From feasibility research we have concluded that this task can and should be handled by the web application and not by the server API. This is to ensure maximized performance while navigating the images.

The system will also have buttons or switches on the GUI that correspond to certain icons for publication, such as a north arrow. These requirements will use metadata from ISIS to calculate and display the necessary data in a way that is acceptable for professional research publications. These switches will control what metadata is presented in the final figure and we will need to make every option toggleable so that it can be removed or added freely by the user. The icons that we choose to use for the publication images should be ones that are widely accepted or even standardized. As shown below:



We also presented an idea to the Client that we should include some type of legend so the figures an be more easily understood by readers of the figure. He liked this idea and that has been made part of this icon generator requirement. The users will also have the option of removing the legend if they wish.

Since we will need to export these images in a manner that they can be added to publications, we can use Node.JS libraries for this. With a few more buttons we will give users the capability to export figures in multiple file formats. This can be accomplished with the Openlayer JS library. Similarly, our client have requirements for the extracted metadata. Firstly, the metadata should be presented in standardized prose, and then it must be exported as a text file. The text file will be extremely easy to create using either

the C++ server or the Node.JS frontend assuming we already have the data transfer properly implemented, but the file should include some specific items such as Latitude and longitude of image center, Target body, Date of acquisition, Resolution (m/pixel) of original image, Phase angle (sun-surface-observer angle), Maximum and minimum latitude and longitude, Type of projection (e.g., Mercator versus Equirectangular), Altitude of observer (above surface) ,and the Slant range to center of image. If these are successfully completed, we will achieve milestones in the export process.

While our product should be able to zoom in on images to isolate specific parts of that image we will also need to be able to show the scale of the image or the indicator of the sun. In order to avoid the occurrence of blocking images, stuff like the scale bar and sun indicator should be in the margins of the image, not covering the image data. In terms of the Scale bar, this will be done by the specification in the Scale bar doc. These specifications were given to us by our Client and thus should be followed to a tee. For monochrome images, we decided on including digital number(DN) value bar. On the other hand, for other normal images we decided on including a dynamic scale bar like the one on the Pilot software. This will give users the capability to see the scale of the figure before cropping the image they want. The specifications included the size that the actual scale bar should be, as well as the standardized format for displaying the scale. We discussed with the client that this version of the product would only need to have the the straight scale bar and not the curved scale bar for planetary adjustments. Our client has set the curved scale bar as a stretch goal for us. If we get to this stretch goal more research will have to be done into the mathematical calculations that occur in the background for the curved bar.

Our product needs to be able to take in images from the user. This image is then able to be cropped, have metadata extracted and displayed. The images that will be uploaded, will always be map-projected .cub files. The Image files will never exceed 2GB. This is the primary way the user will communicate with our product.

Section 5: Non-functional requirements

There are 4 key Non-functional requirements that we have outlined for this project. Availability, Speed, Security and must be Intuitive. The product needs to be available on any OS. Namely Unix and Windows systems. The process must be fast and take no longer than 10 minutes. Our Product needs to be secure. We will have our product hosted at the USGS, and it will feature a front-facing web app. This opens up USGS servers to security issues and will require us to make security a primary focus. Finally, our product will need to be intuitive to use. Our primary user base is going to be research scientists who may, or may not have a technical background. Our app's GUI will need to be easy to use by anyone with at least a moderate background in research science.

The first requirement is that our product is available to users on Unix and Windows systems. Currently, the software that will be performing all image editing and extracting metadata (ISIS3), is only runnable on Unix systems. This prevented us from going with an app that is downloaded onto a computer. To get around this, we decided to go with a web app. This makes the user's OS irrelevant, so long as they have a browser and an internet connection.

Our Client required that the entire process, should take no longer than 10 minutes to complete(as a new user). This will not include time to upload an image to the server. Due to the large size of the images(2GB), this may increase the wait time. Interacting with the GUI and getting input should take no longer than 10 minutes.

Our third requirement is that our product is secure. The USGS takes security very seriously and will not permit any software to run on their server that is insecure or opens their servers up to attack. Because of this, we are placing an emphasis on security. There are several node libraries that make applications/ web apps more secure. We also plan to "salt" any input that is given to the server, to meet this requirement.

Finally, our product needs to be intuitive. Our user base is primarily research scientists, that have little to no coding experience. This means we need to tailor the user experience to be usable by anyone. To accomplish this, we are going to test our GUI with several different users at the USGS, and make changes based on their input.

Section 6: Environmental Requirements

Our client only had one environmental requirement for our product. We need to use the ISIS3 software. The software can natively edit .cub files, which are a custom image format used at the USGS. It can extract metadata found within ISIS3 and has a host of functions that use complex physics/ astronomy equations to get their answer. If we decided not to use the ISIS3 software, we would essentially be “remaking the wheel”, and would be spending the majority of our time, finding ways to get metadata from .cub files and looking up physics equations to make into functions, which would then return things like sun orientation, etc.

Section 7: Potential Risks

Every project has its own set of risks that could occur and every implementation of a project will have vastly different risks. Our risks are relatively slim when we examine them from a far but they are important nonetheless. Since we will be hosting this web application on a server that means we have a security risk because we are opening a front facing web application. These risks can be minimized with a sandbox environment or extra security measures for data transfer. Another risk we may observe is a server crash or lack of internet. Since our implementation is using a web application framework if a user does not have access to the Internet or the server goes down the to web application will not run at all. Lastly, a potential risk could be that the web application does not portray the information properly and we are left with misrepresented images.

The most potential risk we see is that a malicious package could get sent over the network to the USGS servers. If someone injected a package into our network from some other access point they could send files to the USGS server. Another reason we are planning to use HTTPS is that of the added security measure it contains. It sends data through encoded byte stream which is much harder to read with a simple network package capture. There is one way to guarantee that no outer USGS files are messed with and that is to run the server in a sandbox environment. As long as we keep our server away from the main USGS network there is no risk of a security breach.

Another risk is that eventually this application gains user base and the server begins the struggle. If the server's processing power is being consumed by ISIS calculations with enough users, the whole server could crash which could lead to data corruption and many angry researchers. This problem is not even present with our current implementation because we will be limiting the number of client connections to the server. Although if this application proves useful and the user base grows large enough it will require a larger server slot.

Lastly, there is a risk of misrepresenting scientific discovery. ISIS does a great job of creating these maps but when we pass it into another application with another code base we could get some image manipulation. The metadata will be correct which is good but working with these large image files could cause errors in the image conversions and that's how the image could change slightly. This will not pose a risk if we can detect when or if it happens and if we could correct it somehow. Although if left unattended to these image conversion errors could lead to a slightly incorrect figure which is unacceptable in a research community.

Section 8: Project Plan

Our planned project can be broken up into the following steps: Server running ISIS, Server running GUI, GUI calling ISIS command, Server returning ISIS results to GUI, Uploading/Editing images and Intuitive GUI. As each of these milestones is hit, we become closer to implementing our MVP.

We plan on starting with our server because every other part of the app relies on it. ISIS and the GUI require the server to be run. The server also relays messages between the Server and ISIS. This is how the user will be able to run ISIS commands on the fly, and have the results of that command displayed back to the user. This means after the server is complete, the next two steps are getting ISIS to run in the background and having a basic GUI hosted by that server. Once this is complete, the next step is communication between ISIS and the GUI. This will involve the GUI sending a message to the server, the server will then recognize it as an ISIS command request. The Server will then send that command to ISIS with any relevant arguments. It then gets the results of that function and sends the results back to the GUI. The GUI then recognizes the response message, and displays that data. Once we can send basic messages between the GUI and ISIS, we then will move on to the major aspect of the GUI, which is uploading an image, and sending it over the server to ISIS. Finally, once all of the functionality is taken care of we will move on to making the GUI more user-friendly and aesthetic. This is the lowest priority part of the project, but will probably take the most time. This is due to the number of changes our client will likely want to make to it. We will allow ourselves at least the last month of the semester to devote entirely to the GUI.

Section 9: Conclusion

Finally, we will make a summary for this "Requirements Specification." To begin with, we will address the importance of the problem and the project, as well as describing issues that our client are facing.

Currently, for scientists in USGS, the process to edit a single publication image could take anywhere upto 2 hours, making great figures very difficult to add to research papers quickly. Our client - Dr. Laszlo Kestay, who is a research geologist at the USGS has used this software for a long time and he wants to make a great improvement on it. We all recognize that the USGS is doing a good job in integrated System for Imagers and Spectrometers(ISIS). It's only drawback is that the slow speed to process images and data. So, out team - Orion, including three members will be responsible to fix it perfectly. Our work on this project will be very important and needed if we want to increase the speed of exploration and discovery, and our product will be used by thousands of scientists all around the world.

In short, to solve this problem, we plan to use a web application with a C++ backend. Firstly, we must run the ISIS software on a Linux based server. Our product will allow users to upload image files to a web application and receive .cub of the image from the server. This is the most basic and important interaction that our web application needs to be considered successful. Our web application then needs to be able to navigate the image to find the desired portion of the image and then crop it out. This part will be processed by the front end. Our products need to have buttons or switches on the GUI that represent the specific metadata portion to be included in the publication image. Finally, our product will allow users to export the image and metadata in format that can be pasted into publications. The image can be in any format that supports text images such as JPG, PNG or PDF, and the metadata must be exported as a text file which includes some specific items.

In this requirement document, the most important thing we have done is we summarize a clear thinking for the overall project. In the functional requirements part, we discussed how the users will be expected to interact with the system and why we want it that way. In the non-functional requirements part, We have outlined four key non-functional requirements that we need: Availability, Speed, Security and must be Intuitive. We also talk about the only one environmental requirement for our product, that is we need to use the ISIS3 software because it can natively edit .cub files.

In the project plan, We described a number of milestones in terms of the functional requirements for the system and we laid out when these milestones will take place in the months to come. We plan to start with our server because the rest of the application all depends on it. After the server is complete, our next two steps are to have ISIS running in the background and have the basic GUI hosted by the server. As I said above, we have a clear plan for our product. From server running ISIS, to finally having a intuitive GUI for users to upload/edit images, we will complete our product step by step. Although in the process we will use a lot of techniques such as Resting API, Node JS libraries and so on, we are confident to overcome all technical difficulties and to perfectly complete our web app.

Glossary

1. GUI = Graphical User Interface is the mechanism at which a user can interact with our application

2. MVP = Minimum Viable Product is a application that has just enough features to satisfy early clients

3. MOC = Meta-Object Compiler, these files are used by ISIS in order to create the .cub files

4. HTTPS = Hypertext Transfer Protocol Secure

Appendices

- Scale Bar Spes: https://ngmdb.usgs.gov/fgdc_gds/geolsymstd/fgdc-geolsym-sec35.pdf.