Software Design

Version: 1.1

Team: Orion
Sponsor: USGS
2/8/19

Isaac Shaffer

Brandon Kindrick
Chadd Frasier
Yuxuan Zhu

Table of Contents

Introduction:

Planetary science is laying the foundation for humanity to become a space-faring species by discovering more about the geography of our neighboring planets. In recent decades, The National Aeronautics and Space Administration (NASA) has achieved more advancement in the field of planetary science than anyone could have predicted. What we can't ignore is the fact that the United States Geological Survey (USGS) has also played a vital role in this development. NASA works closely with the USGS to meet all of their data processing and research needs. The USGS has spent more than 30 years developing the Integrated System for Imagers and Spectrometers (ISIS) to decode and examine these massive binary files from NASA's imagers. ISIS decodes these files and retrieves the images and metadata that goes along with them. Then using the metadata ISIS can be used to for geospatial research. These precise topographic maps produced by ISIS are used every time astronauts look for new potentially landing site on the Moon, Mars or even farther planetary bodies. The USGS's contribution to space exploration is the accurate geospatial maps, and without them you could say astronauts are literally lost in space.

We are team Orion and we have been given the opportunity to work with the USGS on the Planetary Image Caption Writing Project. This project is about creating an easier interface to utilize the power of ISIS for users of any experience level. Our client Dr. Laszlo Kestay is a Research Geologist at the USGS and he is asking for a more usable viewer tool for the ISIS software. One of his jobs is to prepare images for publications by highlighting important aspects of the image and including important icons to the image like a north arrow or a sun icon to denote where the sun was at the time the image was taken. In order to achieve this Dr. Kestay must load the same image into multiple processing softwares. The main commands of ISIS are currently on command line with some very simple image viewing interfaces, and our team's plan will be to take ISIS and put it into a web application.

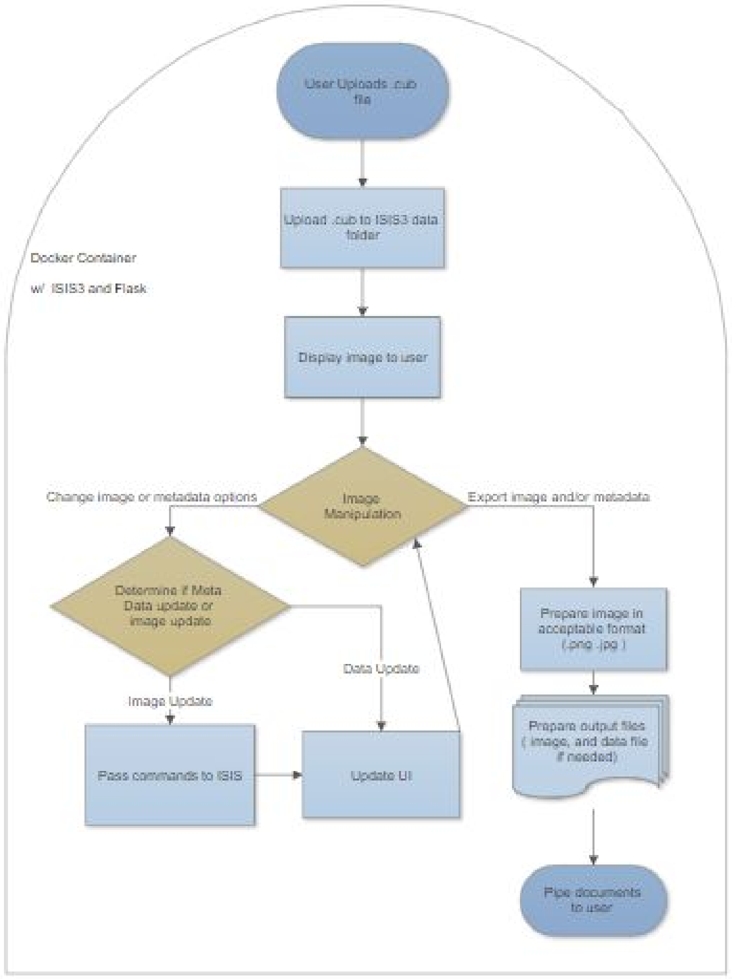
Our team decided that creating a web application that utilizes ISIS for the complex calculations is the best solution for our client because it will give any user access to ISIS no matter what operating system they are using. As of now ISIS can only be installed on Linux or MacOS, and using a web application will give any user the freedom to utilize ISIS without requiring a full install of the software. Our web application must be able to add standardized graphical symbols for specified geospatial information such as a scale bar. Our application must also be able to retrieve additional geospatial information in a subsection of the image. It must also be capable of extract and export the data in an easy to read format(s). Lastly, the application must be able to export the user's image with all the added icons and data pieces. As for the performance of our application, Dr. Kestay wants this whole editing process to take roughly ten minutes for a single image. The only restraint here is that if the user is uploading a very large image it could take more time to process the image. Even with the file variations, on average a user should spend ten minutes or less using the application. In order to minimize load times for the server we are limiting the image files to 2GB, this is because we want to have a proof of concept before attempting it with larger images.

In conclusion, our solution is to create a web application that communicates with ISIS in a way that all the command line functions are hidden from the user. This solution will achieve our stretch goal of bringing the power of ISIS to Windows user while at the same time creating a simple to use interface for photo editing. This solution will create a better user experience because people will not be required to install ISIS onto their machines. That means that anyone can use ISIS without having to follow and prepare the environment manually. This project is focused on the user experience of ISIS and a web application will create the best experience for new and seasoned ISIS users.

Implementation Overview:

We have decided that a Flask based Web Application will be the best solution to our client's needs because of the flexibility and performance that Flask has to offer. Our application will consist of two main portions; the Flask frontend, where users will give input and interact with buttons and images, and then the ISIS3 backend that will be performing all of the calculations on the files. In order to make this application secure we will be developing it inside of a single Docker container. Docker is a lightweight, fast and extremely portable standardized environment that allows for development of web applications such as ours on a local environment. We decided that Flask would be the best option for the user interface because it is smaller and more customizable than Django. The main reason we chose Flask was for the fact that it is a micro-framework for web applications that also us to have a local server for testing. Django is a much larger framework and the extra size is unnecessary for our application and could even be detrimental to the docker container's performance. Flask comes equipped with tools to streamline the website abilities such as handling the Hypertext Transfer Protocol (HTTP) requests. We will be using HTTPS because this is the most widely accepted protocol and it has superior security capabilities. The template system in Flask will create efficient html pages and can even handle style sheets. Our application will be working off a client-server architecture where the clients can connect to a server that is running our Flask frontend. All the data from the user will be forwarded to the ISIS3 portion of the container and then returned to the client. We decided that ISIS3 will be run on the backend because doing this will allow us to use the ISIS command line functions from the Flask application. The reason we chose the route that relies on ISIS3 is because it comes equipped with all the functions needed to manipulate .cub files, which are the files that will be accepted as input from the user. The approach outlined above will yield a secure, standardized, and portable interface for the ISIS3 image manipulation software is a way that simple to use and will not require an install of the software.

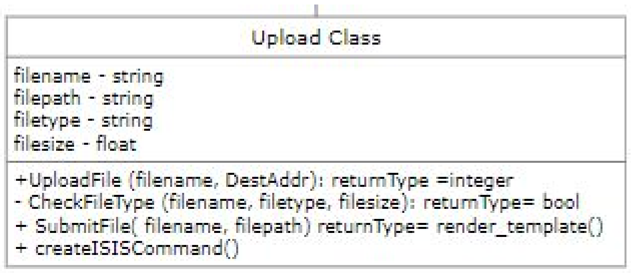
Architectural Overview:


User Uploads .cub file
Upload .cub to ISIS3 data folder
Docker Container
w/ ISIS3 and Flask
Display image to user
Image Manipulation
Change image or metadata options
Export image and/or metadata
Determine if Meta Data update or image update
Data Update
Image Update
Pass commands to ISIS
Update UI
Prepare image in acceptable format (.png .jpg)
Prepare output files (image, and data file if needed)
Pipe documents to user

Starting from the beginning of our architecture, the user will be prompted to upload a .cub file to set the whole process in motion. The system will be required to validate the file type of the upload before posting it to ISIS3, and if the file validation throws a flag it will prompt the user to submit a new file. If the file is validated for a proper file type, the file will be passed to ISIS3 to be processed and stored for later access. In order to more easily manipulate the image from ISIS3 we will convert it to a more standardized image to allow for easier importation into Flask. After we have the image and metadata prepared we will use the Flask HTTP resources to post the new image and data to the web app. After the image is properly displayed the user will be able to crop, highlight, and add icons to the image. There has to be a decision point here because the user will have the freedom to export the image as soon as it has been properly uploaded. If the user chooses to manipulate the image we must decide what type of manipulation it is. If it is a data involved manipulation we can add icons or bars using the metadata. If it is a photo manipulation such as cropping we must pass the file and pixel locations to the cropping function on ISIS3 and return the newly cropped image and metadata using a template and passing it the new image file. If the user applies both image and data changes, we must manipulate the image before the data because the metadata will change after the image is cropped. After we display the newly updated web app we again will allow the user to further manipulate the image or the user can begin the exportation process. When the user starts exporting they must begin with a simple interface that configures the save file path and type. After the user selects an appropriate file type and path they can click a button to begin the export. We will then need to prepare the output files using functions that read the users output settings and prepares either the images or the metadata or both into proper file types.

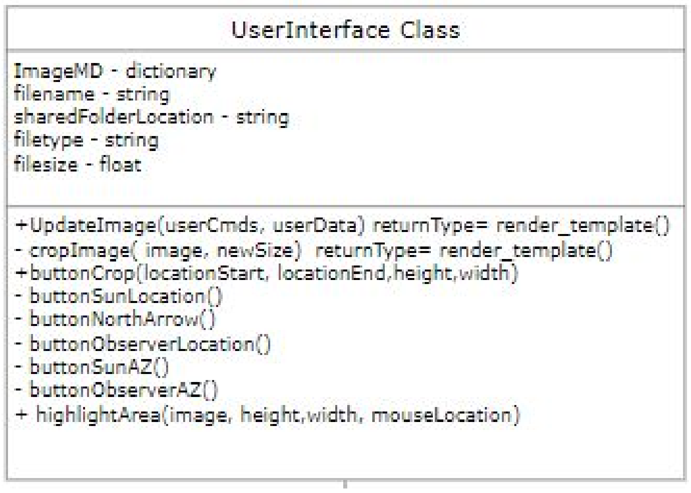
Module and Interface Descriptions

The system will begin with an Upload class that will be responsible for informing the user to upload an ISIS3 cube file and present a window interface for the file upload. We will achieve this by creating an UploadFile function that will utilize a helper function to verify the user's input. We will only be accepting cube files and if this function does not pass then the user will be prompted to retry a new file. After the file is verified we will pass the file to a create Isis Command function that will format a sting that can be passed to an ISIS3 decoding function. After we create the command string we will submit it to the ISIS3 portion of the docker container where a function can read the command in from the frontend and perform the necessary commands in the ISIS software. The main interface will receive the image and metadata from ISIS and store it on the web app for better access. The upload class will also be responsible for configuring the ISIS API and shared folder locations of the application. After a successful upload, we can parse through the cube metadata and retrieve the instrument that is currently being used, the commands that that instrument uses, and all the tags that are acceptable for the camera type. This will act as a configuration for the command side of ISIS3 that will make it easier to validate the current commands being passed.



```
Upload Class
filename - string
filepath - string
filetype - string
filesize - float
+UploadFile (filename, DestAddr): returnType =integer
- CheckFileType (filename, filetype, filesize): returnType= bool
+ SubmitFile( filename, filepath) returnType= render_template()
+ createISISCommand()
```

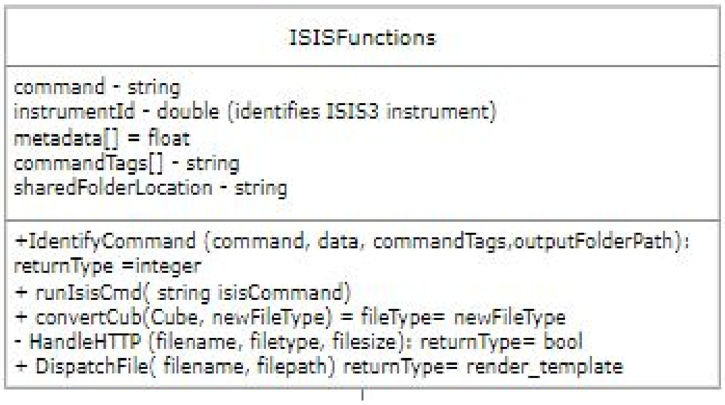
The user interface class will be responsible for all of the user interactions with ISIS. The user interface will consist of check boxes and buttons to add and remove icons from the image as well as a large photo view. The check boxes will work wonderfully for the metadata section because the user can simply check which data they would like included in the figure which is stored locally. Buttons will better for things like adding icons to the image because ISIS can help us calculate where it will need to appear. So we will be using buttons to apply image changes and icons. The cropping feature for example will work in a similar manner to the cropping features in most word processors. A button with scissors on it that will change color or look in some way when clicked to denote that



```
UserInterface Class
ImageMD - dictionary
filename - string
sharedFolderLocation - string
filetype - string
filesize - float
+UpdateImage(userCmds, userData) returnType= render_template()
- cropImage( image, newSize)  returnType= render_template()
+buttonCrop(locationStart, locationEnd,height,width)
- buttonSunLocation()
- buttonNorthArrow()
- buttonObserverLocation()
- buttonSunAZ()
- buttonObserverAZ()
+ highlightArea(image, height,width, mouseLocation)
```

it is in cropping mode. When cropping we will track the mouse location and allow the user to create a rectangle outline on the image that will be the newly cropped image. Four coordinates will be sent to ISIS. In order to do this we will need to send the data about the pixel locations of the start of the box and the dimensions of the box to the cropping function in ISIS3. We will need to validate every aspect of the the commands before sending it to ISIS3. We will associate buttons of the UI with batches of commands for ISIS3. For example the cropping feature would take the input and from the user in the form of a outlined rectangle we will save the size of the rectangle, the location of the top left corner of the box in the form of pixels away from the edge. We will then want to use the isis2std function to change the image format to a png that can be very easily displayed onto the web app. We will need these types of functions for adding every icon including the Sun azimuthal direction, a north arrow indicator and the observer location. The user interface will also have a function that will add highlighted areas to the image to help the user bring attention to portions of the image.

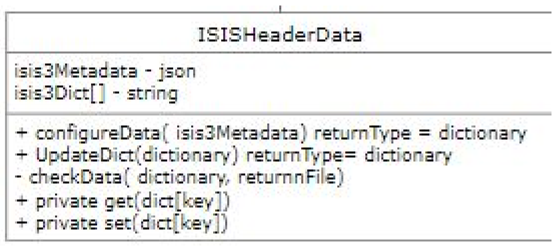
The ISIS control functions will be invoked by the buttons on the user interface and these functions will act as a restful api to perform the necessary actions on the cube file and return the new image. The api will need to verify each piece of the command that it receives and perform commands in the correct order when required to do so. After changing a cube, ISIS3 will need to create an image that Flask can display. Using an api to manage these processes will create a fast and easily maintainable interaction process between the web app and ISIS. The api will handle all the commands and return the image by running a conversion function on the image and icons to be included. Writing a simple api in Flask will create a very simple and fast HTTP handling system. We will need to run a function that sends the new data back to the client after the functions are run and the Flask GET and POST functionality will make this job simple.


```
ISISFunctions

command - string
instrumentId - double (identifies ISIS3 instrument)
metadata[] = float
commandTags[] - string
sharedFolderLocation - string

+IdentifyCommand (command, data, commandTags,outputFolderPath):
returnType =integer
+ runIsisCmd( string isisCommand)
+ convertCub(Cube, newFileType) = fileType= newFileType
- HandleHTTP (filename, filetype, filesize): returnType= bool
+ DispatchFile( filename, filepath) returnType= render_template
```

We will create a class that is devoted to storing and retrieving the ISIS header data that our application requires. It will be easier to efficiently store this data by creating a configuration function that will convert the ISIS header data from a json format to a python dictionary. We will use a dictionary to take advantage of the key-value format that python is capable of. Using a dictionary in this case will allow us to access any piece of metadata with the exact same variable name and a unique key. After the data is properly configured the ISIS header data class will be responsible for



ISISHeaderData

isis3Metadata - json
isis3Dict[] - string

+ configureData(isis3Metadata) returnType = dictionary
+ UpdateDict(dictionary) returnType= dictionary
- checkData(dictionary, returnnFile)
+ private get(dict[key])
+ private set(dict[key])

updating the dictionary if the values are changed, either by cropping or by rotating. This will require two functions, one for checking to see if the data new data is different than the current dictionary and a function to overwrite the dictionary if need be. In order to test and change the values we will need to write a generic get and set method that reads the input in an effective way so that we do not need more than one function for the data. The getter function will return the value of the key it is given, this means that we will need to validate the output before passing it to other functions or comparing it against other values. The setter function will only be called in the update function to prevent from accidental overwrite.

Implementation Plan

In this section, we are going to provide a design-centric implementation timeline for our project. As required; we designed a gantt chart to reflect each required task and its responsible member(s). Using a gantt chart helps to plan, coordinate, and track specific tasks in our project.The following is our team's current gantt chart:

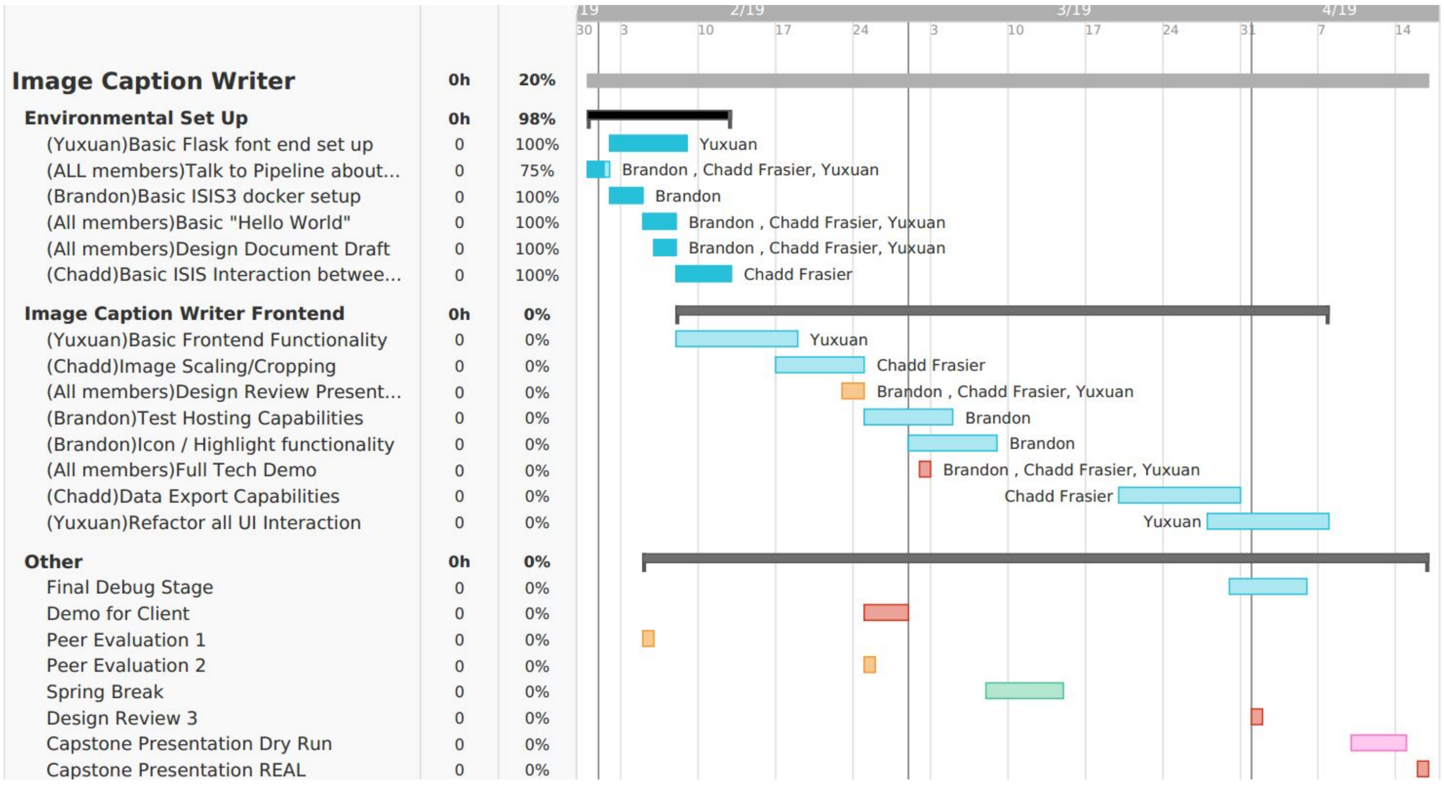


Image Caption Writer	**0h**	**20%**
Environmental Set Up	**0h**	**98%**
(Yuxuan)Basic Flask font end set up	0	100%
(ALL members)Talk to Pipeline about...	0	75%
(Brandon)Basic ISIS3 docker setup	0	100%
(All members)Basic "Hello World"	0	100%
(All members)Design Document Draft	0	100%
(Chadd)Basic ISIS Interaction betwee...	0	100%
Image Caption Writer Frontend	**0h**	**0%**
(Yuxuan)Basic Frontend Functionality	0	0%
(Chadd)Image Scaling/Cropping	0	0%
(All members)Design Review Present...	0	0%
(Brandon)Test Hosting Capabilities	0	0%
(Brandon)Icon / Highlight functionality	0	0%
(All members)Full Tech Demo	0	0%
(Chadd)Data Export Capabilities	0	0%
(Yuxuan)Refactor all UI Interaction	0	0%
Other	**0h**	**0%**
Final Debug Stage	0	0%
Demo for Client	0	0%
Peer Evaluation 1	0	0%
Peer Evaluation 2	0	0%
Spring Break	0	0%
Design Review 3	0	0%
Capstone Presentation Dry Run	0	0%
Capstone Presentation REAL	0	0%

2/19
3/19
4/19
30
3
10
17
24
3
10
17
24
31
7
14
Yuxuan
Brandon , Chadd Frasier, Yuxuan
Brandon
Brandon , Chadd Frasier, Yuxuan
Brandon , Chadd Frasier, Yuxuan
Chadd Frasier
Yuxuan
Chadd Frasier
Brandon , Chadd Frasier, Yuxuan
Brandon
Brandon
Brandon , Chadd Frasier, Yuxuan
Chadd Frasier
Yuxuan

In the Gantt chart shown above, we have provided a detailed schedule of each module in our project. Each task is assigned to a specific team member who will be responsible for that task.

The first huge task we are working on is the Environmental setup. This is the basis of our mission and includes several modules. As we mentioned in "Implementation Overview," we will use Flask for our front end, and ISIS3 on our backend to perform all of the calculations needed. Both the front end and back end are included in our docker container, ensuring security of our app and availability on most OS platforms. The USGS has created a Docker container image on Github that provides everything needed to run our web application: code, runtime, system tools, system libraries and settings. Because all of our members are new to docker, Our first task is to setup our docker environment and establish basic ISIS/Flask communication. We have installed the ISIS docker image on our laptops and installed Flask inside the docker container. The next step is to build a basic communication messaging system between flask and ISIS3 using Flask to call ISIS commands at runtime.

We have assigned specific responsibilities to each member in each module. Yuxuan is primarily responsible for our Flask front end. Chadd will work on the back end with Brandon. Chadd will build ISIS3 interaction and Brandon will be the Docker designer. During this time, we also met with another capstone team - Pipeline, who is working on USGS as well, and are also building an ISIS backend which is also hosted on a Docker container. The communication with them so far has given us insight on how to use Docker and ISIS in tandem.

Our second major task, as well as our full tech demo step, is “Image Caption Writer Frontend.” We will build basic frontend functionality which includes all the features our customer required. The main features will include cropping the image, adding standardized graphical symbols for specified geospatial information, retrieving additional geospatial information in a subsection, extracting the data in a readable format, and finally exporting the user’s image with all the added icons and data pieces. After implementing these functions, we scheduled to test hosting capabilities for our web tool, because we use a docker container and a flask server inside, we must make sure all these things work and will have a long life.

Finally, in the “other” section, includes two very important tasks. They are debugging our web application and showing it to our client. We will need to ensure that Dr. Kestay is comfortable with our installation process and with the functionality of our UI. This will happen in mid April.

Conclusion

After introducing our software in detail before, we will move on to the final section of this design document- conclusion. To give a brief conclusion on our team, our project and our client, we are team Orion, and we are working with the United States Geological Survey (USGS) on the Planetary Image Caption Writing Project. Our client is Dr. Laszlo Kestay, who is a Research Geologist who has worked at the USGS for many years. It is well known that images in space publications are at the heart of sharing information and gaining public interest. The problem our client is facing now is having no convenient way to extract the metadata contained in images other than by ISIS3 command line calls and a pencil. Dr. Laszlo Kestay has to manually extract the metadata by using the binary editor on the ISIS3 file and then type into a document. If he wants to build annotations on the image, he has to use Photoshop. The whole procedure takes a long time and is incredibly tedious. Dr. Kestay is tired of this process and looks to us to tackle this issue.

To solve this problem, Our team plans to take ISIS and make it accessible on any platform via a web tool. It will include a user friendly “viewer tool” on the front end. Our goal is to make the app accessible to anyone whether they are an experienced programmer or someone who has never touched a programming language. We will use the Flask framework to develop our front-end that allows users to process an image, edit metadata, and finally, export the user’s image with all the added icons and metadata. For the backend, we will use ISIS3 because it already has all of the functions we want for our web tool. It will process requests sent by front-end and run corresponding commands then send the result back to the user side. We will develop this web application inside of a single Docker container. Docker allows our team to package up our web application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. As a result, our web application will run on any machine, regardless of OS installation.

In “Architectural Overview” section, we have described the process of the overall operation of our web application. We have provided a graphical display for the whole architecture in that section. From users uploading a .cub file to be stored on the ISIS backend, to users manipulating the image, and finally users beginning the exportation process. We have shown a detailed overview of our architecture and its functionality.

In “Module and Interface Descriptions”, we have provided a design description consisting of UML a class diagram and text description. We showed the functions we plan to implement and then gave corresponding explanations. Finally, on “Implementation Plan”, we used a Gantt chart to show our estimation for when work on the implementation of each module, it's testing, and it’s integration with other components, will be completed. This section also outlines who is responsible or each part of the implementation.

Through this document, we have provided our readers and client with a big picture of our product. Following our schedule reflected in our Gantt Chart, and by following the design of our project, we are confident we can provide the product that will satisfy our client’s requirements and better the production of publication ready image creation.