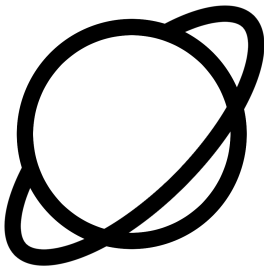


TSO

Progress Report 2

Sebastian Lopez
Gustavo Andres Murcia
Kirk Vander Ploeg
Elijah Ward

T | *Table of Contents*

1 | *Project Description*

Summary

The following is the second of four reports on the progress of our chosen capstone project: ***Telescope Scheduling Optimization***. This project's initial goals are outlined in our first deliverable, and although some things have changed since then, the main work remains to develop a Telescope Scheduling Optimizer system for the Maunakea Spectroscopic Explorer (MSE) herein referred to as **TSO**. The members of our team are Elijah Ward, Gustavo Murcia, Kirk Vander Ploeg and Sebastian Lopez. This project is being supervised by Professor Pauline Barmby.

By working on a weekly basis internally as a team, with Pauline, and having contact with an MSE Database Specialist, we have been able to further define an Overall System Design, Quality Assurance Roadmap, and ongoing Implementation efforts. Previous artifacts introduced in our first deliverable like the Functional Requirements, and various Project Planning have been accordingly updated to reflect the current status towards the completion of the TSO system.

Problem Statement

The Canada-France-Hawaii Telescope (CFHT) team is currently operating using a suboptimal method of producing an observation schedule for researchers that wish to use their telescope for their work. The current method is a very manual and labour-intensive process. Administrators must collect all applications, assess the priority of each of the studies associated with the request, consider a set of **fixed** and **dynamic** constraints and do their best to create a schedule that is optimal. This approach is prone to human error and subjectivity. Before the construction of the newly proposed telescope, the MSE, the team desires a more automated way to create an optimized schedule based on the priority, fixed and dynamic constraints of each observation request.

As a general introduction of how the system will behave refer to the Context Diagram that describes how the current system works and how TSO will be.

Context Diagram *Found in Appendix*

Stakeholders

The Team

The structure of the TSO team remains the same. We have captured an outline of team member roles and guidelines in the **Team Outline**. This can be found in the appendix.

Professor Pauline Barmby

Pauline and our internal team continue to meet on a near weekly basis to go over the progress that has been made with regards to the TSO system. She has had a hand in facilitating questions regarding the existing system, and aiding us in progressing the work further. Documents such as the Functional Requirements have been looked at by Pauline for guidance and approval.

Maunakea Spectroscopic Explorer Team

Through meeting notes and phone calls occurring between Pauline and the MSE team we have been able to narrow down the specifics of what our ongoing implementation will be directed towards.

MSE Database Specialist

Contact to an MSE Database Specialist / Administrator (DBA) has been facilitated by Pauline. The DBA has been providing us with crucial information regarding the existing system and how TSO will be playing a role in the vision that the MSE team has for TSO.

Team Outline *Found in Appendix*

2 | *System Requirements*

The following document is a collection of requirements that the TSO system will adhere to. Through communications with Pauline, consultation of the existing solution, and our plans for TSO, we have captured these requirements and categorized each one as either a Functional Requirement (FR) or a Quality Requirement (QR). When requirements need modification, they are to be verified by both Pauline and the team.

Changes to Requirements

In comparison to our first deliverable certain major requirements have been clarified and thus some modifications and changes to the functional requirements have occurred. Originally our understanding of the scheduler was that a variety of schedules would be produced, each of which would then be analyzed in order to determine the most appropriate one to use. Some modifications have been made to this functionality as our objective has become clearer through communication with the stakeholders.

Currently a revised vision of the scheduling functionality is as follows. In comparison to the old system used by the CFHT, the main goal of our system is to replicate and automate the remote observer's responsibilities. Thus we will have to implement a scheduler that in real time will gather all the necessary input data through which it will pick the best observation to perform after the current observation has expired. This may entail that our scheduler would have to be running constantly (or frequently enough) such that it will constantly be checking if the schedule currently in use is the most appropriate for the real time conditions.

Another addition made to this second set of requirements, is the functionality of creating long term schedules based off historical data. From our understanding these schedules would act as a foundation on which to build on once the real time data is gatherable. For example if historically the best time to observe celestial body *A* is between 9:00 pm - 11:30 pm during the first two weeks of march a schedule will be created ahead of time that satisfies this data. However come March 1st the weather conditions are unfavourable for observing celestial body *A*, so our system would then gather all the current real time data and from there determine the best alternative ; which could still be to observe the same body *A* or observe another body *B*.

Finally another major shift in the requirements is that the stakeholders made it clear that they want the system to provide detailed feedback. This feedback will inform the user why the current observation block was chosen(and all the respective metrics used to determine this) and most importantly it will specify what other blocks it was compared with. This functionality will be extremely helpful when analysing sets of data and how a change in a certain constraint could lead to an entirely different schedule.

System Requirements *Found in Appendix*

3 | *Project Plan*

We are continuously maintaining the initial documents first drafted in our initial progress report throughout the development of TSO. We have found that our initial instincts to use G-Suite as the host of all of our documents has nurtured positive work internally within our own team and work with our stakeholders.

Changes to Project Plan

The one thing that has caused the most impact the initial plan of the project was our initial exposure to the actual MSE team. We had been given access to a set of meeting notes by Pauline, and they outline the MSE teams aspirations for TSO. Following this, some initial expectations we had of the overall system ended up changing, which in turn resulted in changes to the Project Plan documents. These changes however are not impactful as one might think, as the concepts founding our initial understanding were abstracted from the resulting changes that occured.

Project Tasks *Found in Appendix*
Project Timeline *Found in Appendix*
Project Work Breakdown Structure *Found in Appendix*

Most, if not all, of the things we had envisioned with the TSO system from the perspective of system design remain true. We initially took an abstract approach to thinking about what we were building because we knew that it was bound to change - the initial lack of details (which we now have) caused this. Now that we have a clearer picture of how the TSO system is envisioned by both our own team and our stakeholders, we now have a narrower approach in the design of the system.

Proposed Design

Technologies and Tools

Our decision to use python has remained the same and has proven successful in our own initial use. The prediction of pythons easy-to-use methodology has given way to quick prototyping and infrastructure setup within the system.

From the MSE DBA we been given a database schema that suggests the use of C++ in the existing CFHT system. Our prior suspicion that certain aspects of TSO will need to be written in C++ are still there. The performance of the potentially complex scheduling algorithm and the existing solution call for it.

A point should be made that although the existing system may be using C++, it is in our best interest to have any aspect of our language choice abstracted away from anyone interfacing into TSO.

Python

Our use of Python continues and has solidified itself as our technology of choice. We have also begun looking at PIP (the most popular python package management system) as a vital tool in delivering TSO to our stakeholders. We also continue our plan to use AstroPy within the our system as it's pertinence has not faded. We anticipate that this module can help us by being part of the input for our cost function to identify the most optimal schedule.

Protocol Buffers

Our introduction to the current CFHT Database Schema was done through a schema written in ProtoBuf. The given schema has proven extremely useful as it is shining light towards the existing solutions scale and requirements.

We have been inspired by this technology and are looking to including it as our de facto messaging protocol for inputs and outputs. Using it will mean abstracting many aspects of TSO away from the users and maintaining easy to use code - with the potential benefit of also having a "plug-and-play" result with the existing system. This decision will be determined by our next contact with the MSE DBA.

User Interface

After direct communication with the CFHT team, it has been determined that the most appropriate way to interact with the TSO will be through a command line interface (TSO-CLI). The small number of users expected to use the scheduler will consists of individuals who are comfortable using a command line tool. Through this interface, a user can track the observation data that the scheduler is currently using and optionally, the user can add new data into the system. With this information a user can override existing data to be considered for scheduling. Once satisfied with input data, a user can specify the duration of a schedule to be created. This will range from the next best observation to weekly, monthly, and semester long schedules. Once a schedule has been optimized, a report will be returned to the user. This report is to consist of input data and the resulting schedule. The report will also give the user the ability to review other observations that the TSO considered but did not choose to be optimal. The user must also provide remote observers with a schedule once it is created. For this purpose, a schedule will be exported in various forms that are transportable and easily legible for observers (Excel, pdf, etc.).

On additional point to make about the TSO-CLI is that it will enable us to build the application moving forward without becoming blocked by required work on the GUI component. This has two added benefits that will improve our work going forward. Smoke Testing becomes easier in the beginning as a GUI does not have to run at all for us to see instant feedback on our work. Further, if we implement the TSO-CLI to make use of the Command pattern where the actions the TSO-CLI can perform become implementations of Command<>, we will eventually have a plug-and-play situation where any future GUI can easily connect to. We are aiming to decouple the actions the TSO system makes away from any interface we add.

Conceptual Context

In very simple terms, what we are trying to achieve is to output the best possible schedule taking into account what celestial bodies will be observed and all the necessary data for said observation. Our current approach will deal with dividing each schedule into manageable units known as observation blocks. These Blocks will then be grouped together to create a schedule which will be executed and evaluated through the formulating of a **cost heuristic** that will evaluate the candidate schedule's effectiveness.

Each observation block will have a number of constraints that will influence its placement within a schedule. So far each observation block consists of the following constraints: Fixed; dictates whether the observation block will be used for a long or short period of time in order to gather all the data necessary. Dynamic; unpredictable conditions such as the weather will definitely be a limitation that will need to be accounted for. Priority; the observation of certain celestial bodies may be more important than others.

A schedule will then be produced as a combination of observation blocks, and then our job will be to measure the effectiveness of each respective schedule. Currently, the metrics used to measure the effectiveness of a schedule are completion percentage and quality of observation during each observation block. During a given observation block it may not be possible to observe the target body for the entire period of time due to dynamic conditions such as the weather. We should be able to measure the success of our generated schedules using some predefined metrics. We can build a "completion" metric by assessing the progress made towards the completion of all necessary observations for the project during that observation block, reported by the observing researchers. We can also discern a "quality" metric for each observation block by assessing the percentage of achieved progress within the duration of the block, relative to the total estimated time to complete the observation. From these metrics we can qualify the success or shortcomings of our chosen approach, giving the opportunity to those who wish to improve it in the future.

With respect to the more technical aspect of our methodology, we will employ **genetic algorithms** as an optimization method. Through the use of **genetic algorithms**, we hope to find the best configuration of observation blocks considering their respective constraints by generating a population of schedules and iterating to minimize a cost function.

Architecture Context

The TSO system is a sort of *expert system* for the purpose of determining an optimal schedule for observing celestial objects with a telescope. It consumes data from numerous sources in order to make an informed decision in place of a human. To refer to the commonly understood *job-shop problem* (JSP), TSO aims to solve a single-machine job-shop problem due to the fact that the telescope can not observe two objects at once and there is only one telescope.

TSO Flow

Firstly, proprietary information is consumed from the client's data model through the **Input Transformer**. Following this, a collection of **Observation Block** objects are created to encapsulate the data given as input, these form the foundational items TSO will process.

From here, the blocks can be passed as input to the **Scheduler** entity, which will serve as the entrypoint to initiate the generation of the optimal schedule. The **Scheduler** makes use of **ga.py** (**G**enetic **A**lgorithm), a functional python module created by us that depends on the **deap** python library for distributed evolutionary algorithms.The **ga.py** module will make use of a **Cost Heuristic** object which will be used for evaluating the fitness of individual solutions within each population generated by **deap**. The **Cost Heuristic** will consume information from each **Observation Block** in addition to the context provided by it's configuration within the candidate solution.

From here, it can pass the relevant information to the **Constraint Analyzer** to assign a total *cost* (inversely, *fitness*) to the candidate solution. In order to arrive at a *cost* for each solution, the relevant details will be provided to the **Constraint Interpreter**, which will glean further information from integrated libraries and APIs such as **astropy** and a **weather API** in relation to the configuration of the **Observation Block**. This information will be used to construct **Fixed Constraint** and **Dynamic Constraint** objects whose satisfaction will heavily impact the assigned *cost/fitness* of the candidate solution.

As the genetic algorithm necessitates, this sequence will be executed rapidly and for a very large number of iterations. The **ga.py** module will use **deap** to generate a population of *n* candidate solutions, where *n* will be some population size such that increasing it further would give a negligible probability of yielding a better solution. Furthermore, *x* generations will be generated wherein the best candidates from the previous generation are "crossbred" in an attempt to create an even better solution from the best features of the previous generation. For these reasons, the assignment and assessment of constraints will need to be a very efficient process. Optimizations such as proactive fetching of API data will be necessary.

Architectural Diagram *Found in Appendix*
Entity Outline *Found in Appendix*

5 | *Implementation*

Since the delivery of progress report 1, our team has now begun the implementation of the actual TSO system. Due to the ongoing revelation of certain details that will facilitate future work, our current work corresponds to the areas of the application we have the greatest confidence in developing without lost effort.

Workflow

One thing we have diligently been following is our use of Git and it's best practices. The use of branches, pull requests, code reviews is starting strong and will continue to do so once the size of application grows. We have chosen GitHub as our repository host due to all members of the team already having accounts and having experience with their workflow model. In order to maintain consistent development environments among each of our team members, we have decided to use a tool called Conda, which is a package and virtual environment manager for python. The state of our working environment is exported to the git repository and can be easily cloned and duplicated by anyone wishing to contribute to the project.

Packaging

Since we are working with python, we have looked to PIP as a packaging tool to use within TSO. This also leads us to the point of how exactly we will be delivering this system to our stakeholders. If the conversation of implementing a simple UI continues we must define our packaging strategy further once the UI solutioned, but at the moment we must develop our own way of using the various.

TSO-CLI

We know that they indeed have enough computer skills where working with a CLI interface would not be foreign to the CFHT team. So we have begun work on implementing the initial entry to TSO through the TSO-CLI.

Infrastructure

Foundational elements of the TSO have also been built. The following TSO Entities now have implementations and their development continues; Constraint, Cost Heuristic, Dynamic Constraint, Fixed Constraint, Genetic Algorithm, Observation Block, Scheduler, Transformer.

TSO GitHub Repository *Link to Resource*

Message Interfacing

A major development in the implementation of TSO is the format of the existing database schema that the MSE DBA has supplied us. We have been given relevant snippets of a Protocol Buffer file describing the current CFHT database schema. This has resulted into the current development and consideration of the messaging interface that the TSO system will be exposing so MSE staff can easily port it over to their current workflow.

Protocol Buffers

Protocol Buffer is an Interface description language which provides a schema for data that is language-neutral. A proto file can then be compiled into useable classes across a number of supported languages, python and C++ being among them.

The current CFHT Proto Schema provides a good idea of what data is collected from science programs and a schema of the data for which the TSO can consume. For the current purposes of the TSO system, the existing schema is acting as a foundational backbone to the new Proto Schema being developed for TSO.

Due to the highly extendible nature of Protocol buffers, future developers will benefit from this tool when creating a database for programs wishing to observe with the new telescope. If updates are to be made to the schema, the changes can be made in a single, shared proto file which will automatically compile new source code to read and write the structured data. Allowing for schema evolution is of great concern when considering a scheduling system which will draw from a database yet to be built.

gRPC Compliance

Protocol Buffers not only provides a flexible option to model the observation request but can also be easily coupled with gRPC - an extension to Proto which offers syntax to define services that operate on proto messages. As is the case with proto, gRPC compliant proto can be automatically compiled into an HTTP/2 API.

Using gRPC, services are provided to the client as methods that can be called remotely. These methods are implemented on the server side allowing for the remote procedure calls from a client to feel as though they are using a service on the same machine. This contrasts with traditional RESTful message/response systems which are highly interwoven with the HTTP 1.1 protocol, the use of gRCP takes advantage of the faster HTTP/2 protocol.

At the moment it is unsure of the role gRPC will take in the development of the communication protocol that TSO utilizes, as it might prove to be out of scope. Confirmation from the MSE DBA could confirm whether this is the case or not. Regardless of this, the use of gRPC could not only simplify the database/scheduler communication but also allow for new applications to be built on top of the TSO.

As part of the Quality Assurance (QA) work that we will be providing our client, we have devised two front facing documents that will reflect the QA work.

Quality Assurance Roadmap

QA Matrix

The QA matrix is a document that is a rough one-to-one mapping of the Project Tasks to the QA effort that will correspond to each task. At the moment, these tasks vary in magnitude, as some of these map to individual tasks, FRs, QRs, or even entire features. To respond to this varying magnitude, the QA tasks also vary in magnitude. These QA tasks range from complex ***Load/Integration/Performance*** tests, less complex and more isolated ***Unit*** Tests, and rough ***Smoke*** testing required for minimal components like the UI.

QA Implementation

For this system we will primarily be using the ***pytest*** framework. This will allow each member of our team to contribute in a seamless way towards their respective QA task. Other frameworks to aid in the implementation of other more complex QA task may be needed.

QA Report

Following the full implementation of the **QA Matrix** outlined above, it is imperative to capture these tests in a Test Report Document. With the use of **pytest-html** we will be able to produce an easy to access document that both the client and members of our team will have access to.

QA Matrix *Found in Appendix*

Deliverable 3 Roadmap

As we look ahead to deliverable 3 and through to the completion of the project, we feel that at this point we have a firm, detailed understanding of what our stakeholders want to be delivered. As we move forward with the design and implementation of our core features, we have a set of tasks slated to be completed in the immediate future.

The most important thing for our team to accomplish next will be to create a “tracer bullet” of TSO. In the book *The Pragmatic Programmer* by Andrew Hunt and David Thomas, a tracer bullet refers to a minimum viable product (MVP) that demonstrates a “slice” of end to end functionality. There are many benefits to creating this kind of MVP. The first is that it gives the team the opportunity to validate their understanding of the software to be delivered against the mental model of all stakeholders, and receive feedback from them. The second is that it reveals hidden complexities earlier in the project that otherwise are hidden during the design process. This tracer bullet should include the ability to receive input data, generate a population of candidate solutions using **deap**, assign constraints within the configuration of each candidate solution, and produce an “optimal” schedule. For this initial demonstration, we will need to be flexible in what constitutes an “optimal” schedule so that we can focus on validating the top-level functionality with our stakeholders.

In order to reach an MVP, there are some tasks to be completed imminently. The first is that we require the addition of a **Schedule** entity that will be used to generate and store candidate solutions for the genetic algorithm module. This entity will store the relationship between a set of **Observation Blocks** and their assignment to a specific date and time.

Secondly, we will need to integrate the modules we are using to assist in the assignment of **Dynamic Constraints** and **Fixed Constraints**. Currently these include **Astropy** and yet to be specified **WeatherAPI**. It will be useful to do this as soon as possible so that the team can tackle the problem of reduced performance introduced by retrieving information from external sources. Prefetching a set of relevant data may be one such solution to this problem.

Thirdly, we should expose the ability to set any relevant constants for the system through a **configuration file**. This will help the team to easily manipulate the system to produce the best result. It also serves to reduce the risk of delivering a system that does not behave *exactly* as the stakeholder wants. Exposing the control of certain behaviour through configuration will allow the client to be self-serving when they use the system to produce their ideal schedule.

Finally, the team will further investigate what value can be delivered to our stakeholder by making TSO a **gRPC-compliant** service. The team sees this as a value add and will weigh the benefits against the time-to-implement in order to determine whether we should consider the work to be in-scope for the project.

Future Maintainers

With the completion of our first version we are aiming to implement the entire TSO system that we have described above, this version should be fully functional and provide the users with the necessary tools to automate the schedule processing and scheduling selection processes. However there will still be plenty of room for implementing additional functionality that although not necessary for the original version, it would definitely enhance user experience. These additional functionalities and fixes could definitely be taken up by future capstone groups.

Further User Interface GUI

It is clear how our current version of the TSO will rely solely on a CLI, which is no real issue since all current users are proficient and comfortable running a program through the command line. However future versions should look towards implementing a fully functional GUI in order to improve the user friendliness of the TSO. The implementation of a more visual user interface should have no effect whatsoever on the functionality of the rest of the system, yet it would be a welcome change for user experience, making the whole system appear more polished visually. This contributes to a better perception of the software.

Further Interfacing

The main objective of our project is to create a very reliable and accurate scheduler, thus all of our focus will be concentrated towards ensuring we succeed in this. Because of this we won't be able to automate every single step of the entire process. One crucial step that will still be done manually even when our system is implemented is the instrument alignment and adjustment. From the schedules the TSO will produce, an instrument scientist is still needed to adjust the instruments to the specifications suggested by the schedule. Future teams could definitely tackle this section in the hopes of further automating the whole process of observing celestial bodies, this improvement will automatically adjust all the instruments and ensure they're following the suggestion made by the scheduler.

Further Optimization

Like any piece of software, we want the TSO to produce meaningful and correct output and do this in the most optimal manner possible. Our first attempt will definitely focus on making the system as efficient as possible. However future groups that choose to further improve the TSO should look for ways to further optimize the system. By this point the system has already been in use for some time giving these future groups the opportunity to hear from user feedback and collect any necessary data that may aid them in the pursuit of further optimization.

Just like any piece of software, future teams will definitely have to focus on maintaining and managing the latest version of the TSO. This will definitely include minor bug fixes as well as any larger problems that may arise after countless of hours of exposure to the system.

Our Consideration

All of the previous points are areas in which the system could improve further, however none of this would be possible if our system is not future proof. This final point is arguably the most important to keep in mind and it's something every team should work towards, starting with us and passing it on from team to team. In order to ensure that our system will not go out of date and is easy to build upon it is up to us to establish a solid and high standard from the beginning in order to make it easier for future teams to maintain this high level of quality. To achieve this every team should write highly cohesive code making it very easy for future teams to add additional functionality to the system without affecting what has already been implemented.

Consistency is a key attribute to ensuring the longevity of a system, by implementing the system with python and using conda as our environment management system we will already have setup the groundwork for maintaining consistency. Thus we highly recommend future teams to continue using these tools. Additionally it is paramount that each team put a lot of effort in creating clear and proper documentation as this will not only help users to better use the system but most importantly it gives future teams the proper information and tools to fully understand the system. Finally due to the impressive magnitude at which technology advances future teams should always be looking to keep up to date with them and alter the system in ways such that it can assimilate and work properly with these future technologies.

A | *Appendix*

Team Outline

Context Diagram

Architectural Diagram

Entity Outline

System Requirements

Project Tasks

Project Timeline

Project Work Breakdown Structure

QA Matrix

Team Outline

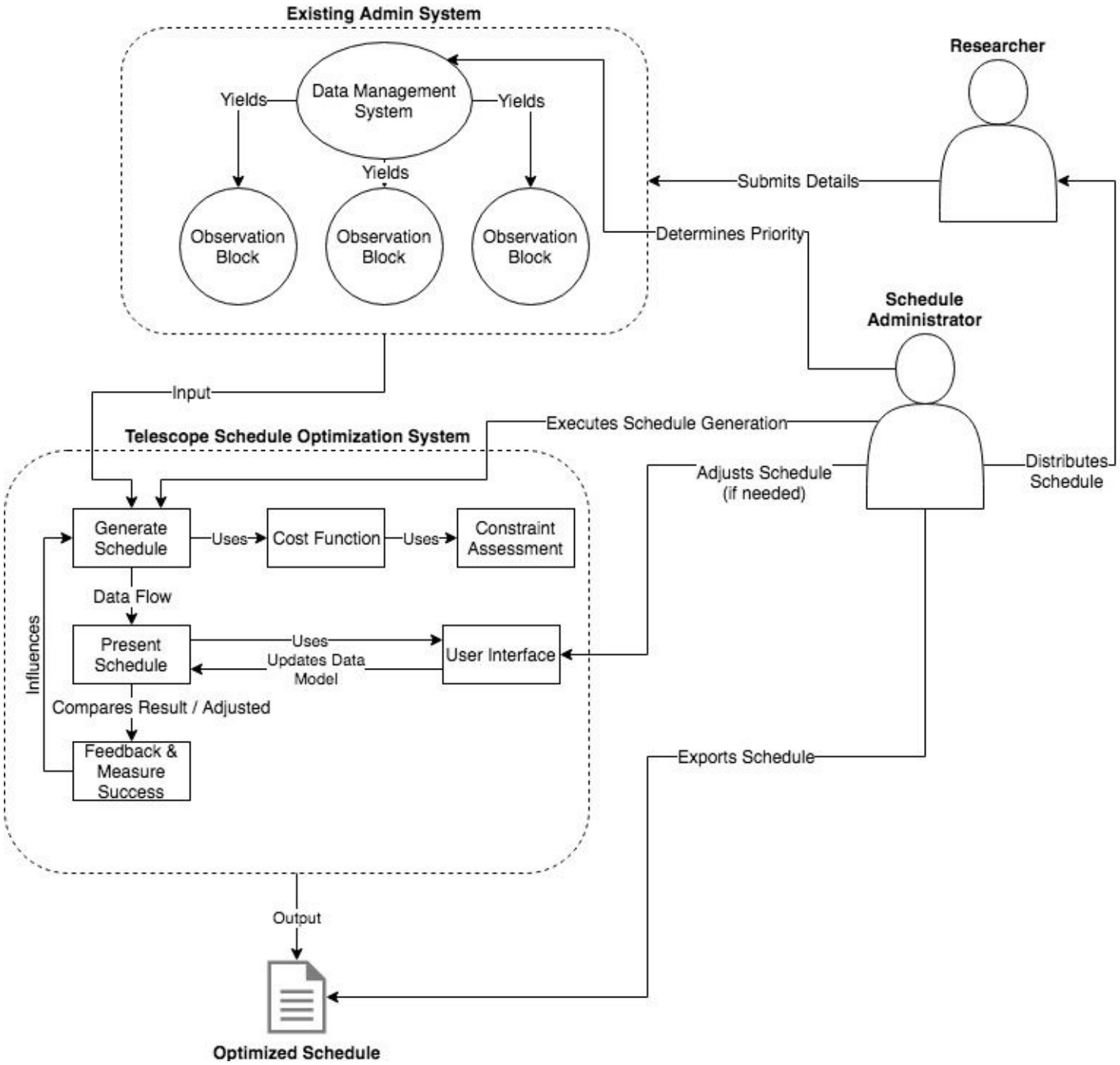
Team Agreements

1. Every team member will serve as a backup to the Project Manager.
    a. Ensures accountability to our deliverables and maintain an open conversation about team expectations.
    b. Ensures that any team member will be prepared to assume PM responsibilities in the absence of the Lead Project Manager.
2. Every team member will be responsible for supporting the Quality Assurance Lead.
    a. Ensures team members create unit tests for their own work
    b. Will be subject to review by the QA lead on completion.

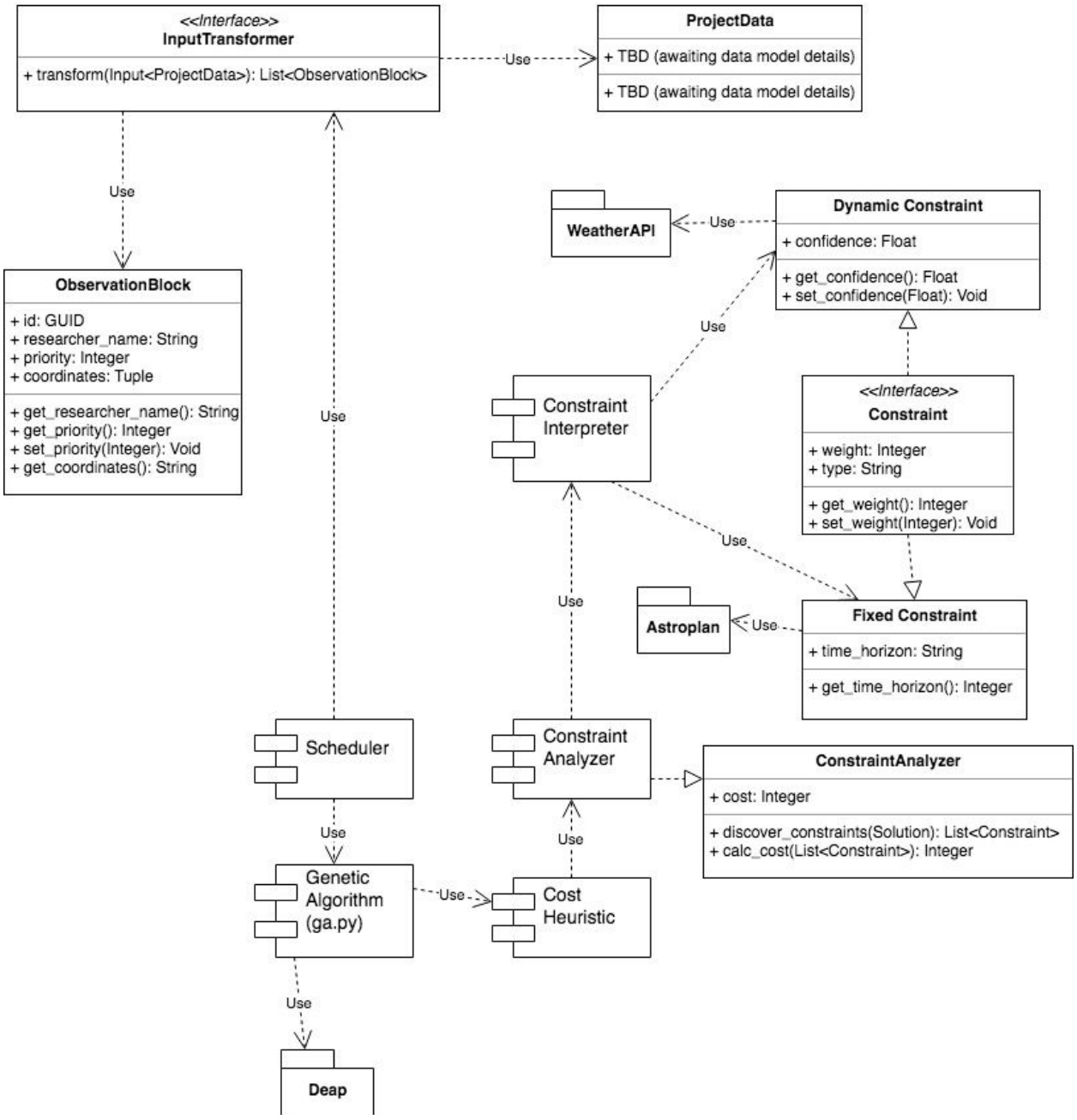
Role Assignment Table

Team Member	Primary Role(s)	Secondary Role(s)
Sebastian Lopez	- Lead Requirements Analyst - Lead Tools Specialist	- Backup Architect
Gustavo Murcia	- Lead UX Design - Lead Project Manager	- Documentation Lead - Backup Solutions & Requirements Analyst
Kirk Vander Ploeg	- Quality Assurance Lead - Integration Testing	- Backup UX Design
Elijah Ward	- Lead Architect	- Backup Tools Specialist - Backup Project Manager

Context Diagram



Architecture Diagram



Entity Outline

Entity	Description
Input Transformer	As the name suggests, the **InputTransformer** object exists for the purpose of transforming the client's input data into our own internal data model so that we can safely decouple the internals of TSO from the data model of the client. By abstracting the interaction with the client's data model through this interface, numerous advantages are gained. The first is that changes to the client's data model will not necessarily cause required changes through the entire system. Simply modifying the **InputTransformer** interface to mirror the changes in the client's data model should be sufficient for most minor changes. The second benefit is that the code can serve as a sort of documentation to demonstrate how concepts in the client's data model map to those within TSO.
Observation Block	The **ObservationBlock** object is intended to encapsulate all of the information pertaining to the act of observing a celestial body for a designated amount of time. In the context of the *job-shop problem*, these objects represent the *jobs* to be scheduled. This is realized using a class module, due to a requirement to have many instance of this entity.
Scheduler	The **Scheduler** object is exposed to the rest of the system through a class module. Although the system will likely only require one instance of this object, it may be useful to construct the object at runtime after feeding it some set of configuration. Therefore, this is to be implemented using a singleton pattern.
Genetic Algorithm	The concept of genetic algorithms will be exposed to the rest of the system through a functional module called **ga.py**. The module will contain a variety of functions that abstract functionality within the Deap library for the purpose of optimization using genetic algorithms. These functions will consume objects and return objects back but they will not need to manage any state of their own and therefore do not need to be object-oriented.
Deap	A popular open-source library for implementing evolutionary algorithms in python. This library will serve as the core of the schedule optimization functionality, but it should be sufficiently abstracted using an interface and dependency injection in order to maintain the flexibility needed to change the library in the future as needed.
Cost Heuristic	The **CostHeuristic** is to be implemented as a class module. It will be used within the **ga.py** module to encapsulate all of the necessary logic to assess the relevant constraints for a candidate solution and assign a *cost/fitness* to the solution based on its satisfaction of each constraint in scope.
Constraint Analyzer	Used by the **CostHeuristic**, the **ConstraintAnalyzer** is a suite of functions that are used to determine if a given constraint is satisfied by the configuration of an **ObservationBlock**. The satisfaction of a constraint will be binary in some cases but we also anticipate some constraints that are satisfied to a partial degree between 0 and 100%, such as with a **DynamicConstraint** like visibility.
Constraint Interpreter	The **ConstraintInterpreter** encapsulates the logic which can take the context surrounding the configuration and placement of an **ObservationBlock** and derive the set of **DynamicConstraints** and **FixedConstraints** that pertain to that block. In order to do so, it will need to use various resources such as **Astropy**, to solve for constraints such as the location of the target in the sky, or a **Weather API** to determine dynamic conditions such as visibility.
Constraints	Constraints exist in two basic variations, **DynamicConstraints** and **FixedConstraints**. The former constitutes constraints that change frequently and are more likely to change between schedule generation and the scheduled observation itself. The later is intended to represent constraints that change very slowly or perhaps not at all. Something like the position of a target in the sky at a given time of day would be an example of a **FixedConstraint**.

MSE:TSO - Requirements

Type	ID	Requirement Title	Status	Category	Requirement	Detail	Related	Notes
Feature	**1**	Data Import	Review	Import, UI	*These are the requirements around the TSOs capability of importing data from external sources.*	The TSO System will require data to be imported for the following types of data. Program Data and Constraint Data		
FR	**1.1**	Program Data Import	Review	Import	TSO's main purpose is to act upon Program data. This data represents the various "programs" that are to be optimally scheduled by the TSO. Imported from a single phase 2-like database.			
FR	**1.1.1**	Input Transfomer	Review	Import	Incoming data from client input will be extracted by the transformer and subsequently adapted to our data mdoel	Ensures consistency across the system and the data imported		
QR	**1.1.2**	Program Data Error Checking	Review	Errors	TSO will ensure that any incoming data will not create data bugs because of the nature of the incoming data.	An example of this could be program data that contains unsupported characters		
FR	**1.2**	Constraint Data Import	Review	Import	In order to schedule Program data, TSO requires data that will determine whether some programs are fit to be scheduled or not.	An example of a constraint is the weather. A Program X could rely on a condition that the atmospheric condition of the night is of a certain value.		
FR	**1.2.1**	Synoptic Observations	Review	Import	The TSO should be able to distinguish when the appropriate times to gather realtime weather information are.			
Feature	**2**	Data Export	Review	Export, UI	*These are the requirements around the TSOs capability of exporting data it processes.*	Key functionality of the TSO system, that allows users to use their results outside of the system itself. Schedules can be exported in a variety of formats, while the data used to create these will also be made available.		
FR	**2.1**	Data Export Action	Review	Export	A user will request TSO to output the final schedule to the desired output format.	Gives the user the freedom to use and view the schedule outside of the TSO.		
FR	**2.1.1**	Data Export Format	Review	Export	A user can export data from TSO as a database (SQL, Mongo...)			Confirm Data Export Format
FR	**2.1.2**	Data Export Format	Review	Export	A user can export data from TSO as an Excel file			Confirm Data Export Format
FR	**2.1.3**	Data Export Format	Review	Export	A user can export data from TSO as a PDF file			Confirm Data Export Format
FR	**2.1.3**	Data Export Format	Review	Export	A user can export data from TSO as a Python Speficic Format			Confirm Data Export Format
FR	**2.1.4**	Feedback Report Export	Review	Export	The feedback report created by the TSO can be exported to the desired format	This funtionality will allow the users to export and save the produced reports which are crucial for gathering information	5.1	
FR	**2.2**	Final Data Visualization	Review	UI	The TSO system will allow a user to view the final schedule the system has determined to be optimal.	The user should be able to view what the schedule currently looks like at any point in time. Since the schedule may be constantly changing an appropriate method of visualization should be chosen.		
FR	**2.3**	Output to MSE	Review	Export	After the TSO has chosen the best observation to perform next, this information needs to be exported in such a format that is consumable by the MSE instruments	This will correspond the requirements confirmed by Pauline and the MSE DBA. It might be required for us to connect directly into a CFHT-like system through some sort of messaging protocol like ProtoBuf		
Feature	**3**	Observation Data	Review	Data, UI	*These are the requirements around the TSOs capability to represent observation data to be processed by the scheduler component of the TSO.*			
FR	**3.1**	Data Identification	Review	Data	The user shall be able to identify the source of the data they have imported or manually entered into TSO	By uniquely identifying the data in the system, no collisions will occur and correct output will be produced		
FR	**3.2**	Data Model	Review	Data	The TSO system will manage it's own data model. It will closely resemble the data model of the import, but could have changes to adapt to the scheduling process used.	An input transfomer will be used to extract the needed information from the client's input data and adapt it to our own internal data model	1.1*	
FR	**3.2.1**	Data View	Review	Data, UI	The user shall be able to view the data that TSO is able to schedule	A UI that displays a common format for Observation Data could be something that will improve the process of selecting data to be scheduled		
FR	**3.2.2**	Data Creation	Review	Data	The user shall be able to add data to the TSO system	This is covered through the Data Import requirements	1.*	
FR	**3.3**	Data Persistence	Review	Data	The TSO system shall persist data between sessions of use	This could either be through a shared database or individual local files on the machines running the software.	2.1.4	
Feature	**4**	Scheduler	Review	Scheduling, UI, Data	*These are the requirements around the TSOs capability to schedule observation data.*	The TSO should produce an optimal schedule taking into account the proposed blocks and the best way to organize these. The output should be presented in a clear, detailed and legible manner.		
FR	**4.1**	Schedule Picker	Review	Scheduling, Data	The Scheduler's main function will be to pick the observation block that is most suited for the current time.	Contrary to our prior understanding, the scheduler should act as a "picker", choosing the most relevant observation to perform next. The schedule chosen is based on input from the database, realtime info (weather) and the realtime data reduction pipeline.	3.*	
FR	**4.1.1**	Next Observation	Review	Scheduling	At any point the TSO should be able to solve and choose what observation to perform next.	The idea is for the Scheduler to be running constantly (if feasible), where it will pick in realtime the most appropriate observation that should be carried out next.		
FR	**4.1.2**	Long Term Observation	Review	Scheduling	The TSO should be able to return schedules outlining an entire night's , week's, month's or even semester's observation schedule.	This functionality would provide a long term "base" schedule. Said schedule would be used as a starting point as it would be based solely on historical data and subsequently modified when the realtime data and information can be accessed.		i.e. a long term schedule would be created and when realtime matches specified time on this long term schedule, then the next observation will be processed using the realtime data.
FR	**4.2**	Genetic Algorithm	Review	Scheduling	The TSO will use an iterative proces of a genetic algorithm using a cost function to determine the best outcome for the final schedule from a selection of N observation blocks	This is a macro level event. This is how the TSO will find the optimal solution		Currently a High Level. Further design is required to determine additonal FRs and QRs
QR	**4.2.2**	Genetic Algorithm Performance	Review	Quality	The TSO's use of genetic algorithms will be tuned to meet speed and quality KPIs set by stakeholders	If the genetic algorithm is tunable, it shall be tunable to paramters that are concrete to the genetic algorithm. For example: One such KPI could be "redundancy" - "As schedulers, we would like to see the next best schedule"		
FR	**4.2.1**	Cost Function	Review	Scheduling	The TSO will utilize a custom cost function to determine the final schedule from a set of N observation blocks	This is a micro level event. This is how the TSO will find the best solution given a set of constraints		Currently a High Level. Further design is required to determine additonal FRs and QRs
Feature	**5**	Schedule Analysis	Review		*These are the requirements around the TSOs capability to analyze the schedules it produces.*	In order to produce the most optimal schedules, the TSO system lets the user choose schedules to analyze and provide an adequate feedback report stating the effectiveness of the schedule.		These Requirements will be filled out when we know exactly what properties of a produced schedule will be analyzed or not. This Req family could easily become something we develop internally
FR	**5.1**	Feedback Report	Review	Quality, Scheduling	Once the appropriate observation has been chosen, the TSO should produce a report containing the input data and the results of the data processing	This functionality informs the user on why the selected observation was the most pertinent in comparisson to the alternatives.		
FR	**5.1.1**	Alternative report	Review	Quality, Scheduling	A major part of the feedback report is to inform the user the alternative observations that were considered	In order to better explain why the current observation block was chosen, it is paramount for the user to know what sets of data were compared in order to better understand why one set was chosen over the others.		
Feature	**6**	Roles	Review	Users, UI	*These are the requirements around the TSOs capability to have different users perform different actions.*	The TSO system will have seperate interfaces for those users seeking to submit a observation block proposal, and a seperate one for the management of proposals, creation and analysis of schedules.		
FR	**6.1**	TSO User	Review	Users, UI	The TSO User will use the system to integrate previously developed systems of data and telescope control into their work stream.	The TSO systems is meant to be used by a small group of people working at the current CFHT site. Because of this, there will not be a need to introduce the idea of a multi-user multi-role system.		
Feature	**7**	Performance	Review		*These are the requirements around the TSOs performance goals.*			
QR	**7.1**	Historical Performance	Review	Scheduling, Quality	If provided with historical Data, the TSO will be able to match the accuracy of previous schedules. In this way, we will be able to verify TSO against the performance of the previous system			

MSE:TSO - Project Tasks

FRs and QR have been mapped to line items to work on. The "Task" type will be a child of the given ID plus it's own ID.

Example: Task 1.1.1 is the first task under the FR/QR 1.1

Legend	
Waiting	W
Done	D
In Progress	IP
Blocked	B

Type	ID	Title	Category	Assignee	Notes	Notes+	Status
	M	Deliverables		ALL	*These tasks indicate the 4 Milestones (aka: Deliverables) that we must submit for the course. Each one represents a unit of work towards Code, Documentation, Client Feedback,..etc*		W
Task	M.1	Milestone 1		ALL	Progress Report 1: 15%, due November 5, 2018		D
Task	M.1.1	Overall project Description		ALL			D
Task	M.1.2	Context Diagram		Elijah			D
Task	M.1.3	Role Matrix		Kirk			D
Task	M.1.4	System Requirements		Sebastian			D
Task	M.1.5	Project Plan		Gustavo			D
Task	M.1.6	Preliminary sketch of the system's WBS		Gustavo	OPTIONAL		D
Task	M.2	Milestone 2		ALL	Progress Report 2: 25%, due December 3, 2017		D
Task	M.2	Milestone 3		ALL	Progress Report 3: 20%, due February 25, 2018		W
Task	M.2	Milestone 4		ALL	FINAL Report and presentation: 40%, Report due April 8, 2018		W
Feature	1	Data Import		ALL	*The Data Import component*		IP
FR	1.1	Program Data Import		ALL			IP
Task	1.1.1	Design Program Data Input Transformer	Design	ALL			IP
Task	1.1.2	Develop Program Data Input Transformer	Dev	Gustavo			W
QR	1.1.2	Error Checking Incoming Data	QA	Kirk			W
Task	1.1.3	Develop Associated UI	UI	Kirk			W
FR	1.2	Constraint Data Import		ALL			IP
Task	1.2.1	Design Constraint Data Import	Design	ALL			IP
Task	1.2.2	Develop Constraint Data Import	Dev	Sebastian			W
Task	1.2.3	Develop Associated UI	UI	Gustavo			W
Feature	2	Data Export		ALL	*The Data Export component*		W
FR	2.1	Data Export Action		ALL			W
Task	2.1.1	Develop Associated UI	UI	Kirk			W
FR	2.1.1	Data Export to Format(s)		ALL			W
Task	2.1.1.2	Design Schedule Consumer for Export(s)	Design	Sebastian			W
Task	2.1.1.2	Develop Schedule Consumer for Export(s)	Dev	Elijah			W
FR	2.2	Final Data Visualization		ALL			W
Task	2.2.1	Develop Associated UI	UI	Gustavo			W
FR	2.3	Output To MSE	UI	ALL			W
Feature	3	Observation Data		ALL	*The Observation Data component*		IP
FR	3.1	Data Identification		ALL			W
Task	3.1.1	Develop interal ID tracker for managed data.	Dev	Elijah			W
FR	3.2	Data Model		ALL			IP
Task	3.2.1	Investigate existing data model from CFHT	Investigate	Elijah			IP
Task	3.2.2	Design TSO Data Model	Design	ALL			IP
Task	3.2.3	Develop TSO Data Model	Dev	Elijah			IP
FR	3.2.1	Data View		ALL			W
Task	3.2.1.1	Develop Simplified UI View of TSO Data	UI	Kirk			W
FR	3.3	Data Persistence		ALL			W
Task	3.3.1	Design strategy for persiting user data.	Design	Sebastian			W
Feature	4	Scheduler		ALL	*The Scheduler component*		IP
FR	4.1	Scheduling Selection		ALL			IP
Task	4.1.1	Design Schedule Structure	Design	Sebastian			IP
Task	4.1.2	Develop Schedule Structure	Dev	Kirk			IP
Task	4.1.3	Design Associated UI to select TSO Data	UI	Gustavo			W
FR	4.2	Genetic Algorithm		ALL			IP
Task	4.2.1	Design Genetic Algorithm	Design	ALL			IP
Task	4.2.2	Develop Genetic Algorithm	Dev	Elijah			W
QR	4.2.2	Optimize implementation to improve KPIs	Optimize	Gustavo			W
FR	4.2.1	Cost Function		ALL			W
Task	4.2.1.1	Design Cost Function	Design	ALL			W
Task	4.2.1.2	Develop Cost Function	Dev	Sebastian			W
Feature	5	Schedule Analysis		ALL	*The Schedule Analysis component*		W
Task	5	Investigate existing Schedulers to determine optimization process	Investigate	Elijah			W
FR	5.1	Feedback Report	Design	ALL			W
FR	5.1.1	Alternative report	Design	ALL			W
Feature	6	Roles		ALL	*The Roles component*		W
FR	6.1	TSO User		ALL			W
Task	6.1.1	Develop system executable delivery method	Dev	Sebastian			W
Feature	7	Performance		ALL	*The Performance component*		W
QR	7.1	Historical Performance		ALL			W
Task	7.1.1	Develop integration test of TSO given historical data	Test	Kirk			W

MSE:TSO - Project Timeline

Legend		
Waiting	W	.
Done	D	;
In Progress	IP	-
Blocked	B	x

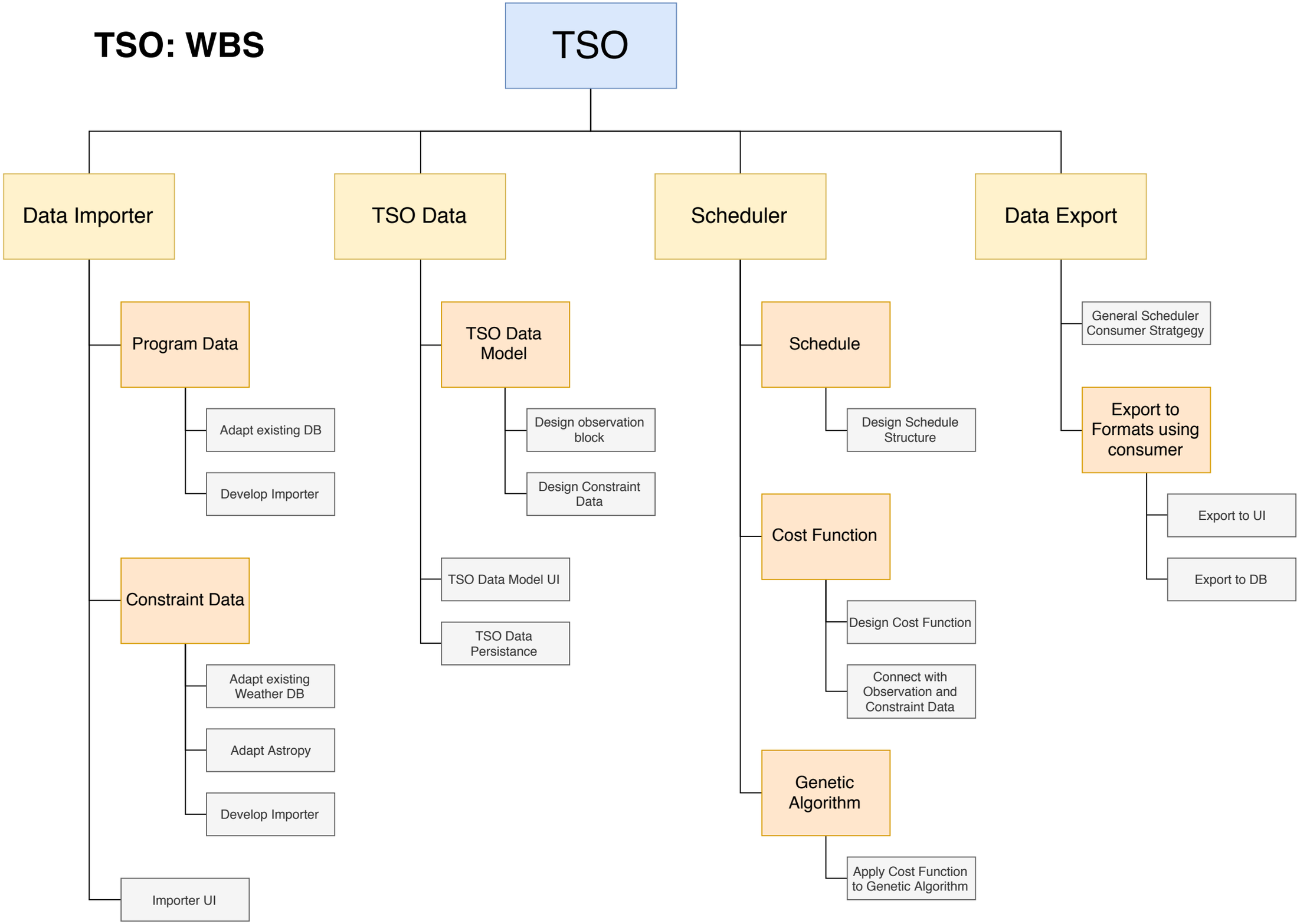
(Column = Week as of Monday)

The status placed in a dark grey header indicates to general timeline of when that high level task is being worked on

CURRENT

	ID	Title	Category	Assignee	Notes	Notes+	Status	Oct-1	Oct-8	Oct-15	Oct-22	Oct-29	Nov-5	Nov-12	Nov-19	Nov-26	Dec-3	Dec-10	Dec-17	Dec-24	Dec-31	Jan-7	Jan-14	Jan-21	Jan-28	Feb-4	Feb-11	Feb-18	Feb-25	Mar-4	Mar-11	Mar-18	Mar-25	Apr-1	Apr-8
	M	Deliverables		ALL	*These tasks indicate the 4 Milestones (aka: Deliverables) that we must submit for the course. Each one represents a unit of work towards Code, Documentation, Client Feedback...etc*		W			;	;	;	;					;											.						.
Task	M.1	Milestone 1		ALL	Progress Report 1: 15%, due November 5, 2018		D						;																						
Task	M.1.1	Overall project Description		ALL			D			;																								
Task	M.1.2	Context Diagram		Elijah			D				;																							
Task	M.1.3	Role Matrix		Kirk			D			;																								
Task	M.1.4	System Requirements		Sebastian			D			;																								
Task	M.1.5	Project Plan		Gustavo			D			;	;	;																						
Task	M.1.6	Preliminary sketch of the system's WBS		Gustavo	OPTIONAL		D					;																						
Task	M.2	Milestone 2		ALL	Progress Report 2: 25%, due December 3, 2018		D										;																	
Task	M.2	Milestone 3		ALL	Progress Report 3: 20%, due February 25, 2018		W																					.						
Task	M.2	Milestone 4		ALL	FINAL Report and presentation: 40%, Report due April 8, 2018		W																											.
Feature	1	Data Import		ALL	*The Data Import component*		IP								-	-	-	.	.	.														
FR	1.1	Program Data Import		ALL			IP								-	-	-	.	.	.														
Task	1.1.1	Design Program Data Input Transformer	Design	ALL			IP								-	-																		
Task	1.1.2	Develop Program Data Input Transformer	Dev	Gustavo			W									.	.	.																
QR	1.1.2	Error Checking Incoming Data	QA	Kirk			W										.	.																
Task	1.1.3	Develop Associated UI	UI	Kirk			W											.	.	.														
FR	1.2	Constraint Data Import		ALL			IP								-	-	-	.	.	.														
Task	1.2.1	Design Constraint Data Import	Design	ALL			IP								-	-	-																	
Task	1.2.2	Develop Constraint Data Import	Dev	Sebastian			W									.	.	.																
Task	1.2.3	Develop Associated UI	UI	Gustavo			W											.	.	.														
Feature	2	Data Export		ALL	*The Data Export component*		W																			.	.	.	.	.	.	.		
FR	2.1	Data Export Action		ALL			W																						.	.	.			
Task	2.1.1	Develop Associated UI	UI	Kirk			W																						.	.	.			
FR	2.1.1	Data Export to Format(s)		ALL			W																			.	.	.	.	.	.			
Task	2.1.1.2	Design Schedule Consumer for Export(s)	Design	Sebastian			W																			.	.	.						
Task	2.1.1.2	Develop Schedule Consumer for Export(s)	Dev	Elijah			W																					.	.	.				
FR	2.2	Final Data Visualization		ALL			W																						.	.	.			
Task	2.2.1	Develop Associated UI	UI	Gustavo			W																							.				
FR	2.3	Output to MSE	Design	ALL			W																							.	.	.		
Feature	3	Observation Data		ALL	*The Observation Data component*		IP									-	-	-	-	-	.	.	.	.	.									
FR	3.1	Data Identification		ALL			W																											
Task	3.1.1	Develop interal ID tracker for managed data.	Dev	Elijah			W																											
FR	3.2	Data Model		ALL			IP									-	-	-	-	-														
Task	3.2.1	Investigate existing data model from CFHT	Investigate	Elijah	***Note:*** *Received* CFHT DB Schema on 26th of Nov		IP									-	-	-	-															
Task	3.2.2	Design TSO Data Model	Design	ALL			IP										-	-	-															
Task	3.2.3	Develop TSO Data Model	Dev	Elijah			IP													-														
FR	3.2.1	Data View		ALL			W													.	.													
Task	3.2.1.1	Develop Simplified UI View of TSO Data	UI	Kirk			W														.													
FR	3.3	Data Persistence		ALL			W															.	.	.	.									
Task	3.3.1	Design strategy for persiting user data.	Design	Sebastian			W															.	.	.	.									
Feature	4	Scheduler		ALL	*The Scheduler component*		IP								-	-	-	-	.	.	.	.	.	.	.	.	.							
FR	4.1	Scheduling Selection		ALL			IP								-	-	-	-																
Task	4.1.1	Design Schedule Structure	Design	Sebastian			IP								-	-	-	-																
Task	4.1.2	Develop Schedule Structure	Dev	Kirk			IP									-	-	-																
Task	4.1.3	Design Associated UI to select TSO Data	UI	Gustavo			W											.	.															
FR	4.2	Genetic Algorithm		ALL			IP								-	-						.	.	.	.	.	.							
Task	4.2.1	Design Genetic Algorithm	Design	ALL			IP								-	-						.	.	.	.	.	.							
Task	4.2.2	Develop Genetic Algorithm	Dev	Elijah			W															.	.	.	.	.	.	.						
QR	4.2.2	Optimize implementation to improve KPIs	Optimize	Gustavo			W																		.	.	.	.						
FR	4.2.1	Cost Function		ALL			W									.	.	.	.	.														
Task	4.2.1.1	Design Cost Function	Design	ALL			W									.	.	.	.															
Task	4.2.1.2	Develop Cost Function	Dev	Sebastian			W											.	.	.														
Feature	5	Schedule Analysis		ALL	*The Schedule Analysis component*		W																			.	.	.	.	.				
Task	5	Investigate existing Schedulers to determine optimization process	Investigate	Elijah			W																			.	.	.	.	.				
FR	5.1	Feedback Report	Design	ALL			W																				.	.	.	.				
FR	5.1.1	Alternative report	Design	ALL			W																				.	.	.	.				
Feature	6	Roles		ALL	*The Roles component*		W																									.	.	
FR	6.1	TSO User		ALL			W																									.	.	
Task	6.1.1	Develop system executable delivery method	Dev	Sebastian			W																									.	.	
Feature	7	Performance		ALL	*The Performance component*		W																							.	.	.	.	
QR	7.1	Historical Performance		ALL			W																							.	.	.	.	
Task	7.1.1	Develop integration test of TSO given historical data	Test	Kirk			W																							.	.	.	.	

TSO: WBS


TSO
Data Importer
TSO Data
Scheduler
Data Export
Program Data
Adapt existing DB
Develop Importer
Constraint Data
Adapt existing Weather DB
Adapt Astropy
Develop Importer
Importer UI
TSO Data Model
Design observation block
Design Constraint Data
TSO Data Model UI
TSO Data Persistance
Schedule
Design Schedule Structure
Cost Function
Design Cost Function
Connect with Observation and Constraint Data
Genetic Algorithm
Apply Cost Function to Genetic Algorithm
General Scheduler Consumer Stratgegy
Export to Formats using consumer
Export to UI
Export to DB

MSE:TSO - QA Matrix

This matrix is a representation of the QA roadmap for MSE:TSO.
It contains an account of all the components of the system and the level of test that will be undertaken for each

Note: Blue rows are the "components" or "units of work" we will be testing for the TSO

Test Type - ID
Integration Test - IT
Load Test - LT
Performance Test - PT
Unit Test - UT
Smoke Test - ST

This describes what things will be isolated as part of the test.
Anything outside of the isolated entities is expected to be mocked

Type	ID	Title	Entity Isolation	Notes	Source
Feature	1	Data Import		*The Data Import component*	test_tbd.py
FR	1.1	Program Data Import			test_tbd.py
UT		Program Data Importer Unit Testing	Classes required in the import	These tests will mock the data source with python mocks	test_tbd.py
IT		Program Data Importer + Connection to mirrored data source	(1) incoming data stream (2) program data importer	(1) will be a Dev mirror of the actual resource to be used in prod.	test_tbd.py
LT		Program Data Importer + Connection to mirrored data source	(1) incoming data stream (2) program data importer	These test will mock a large amount of data coming in through (1) and test the results of (2)	test_tbd.py
QR	1.1.2	Error Checking Incoming Data			test_tbd.py
UT		ProgramDataImportErrorHandler Unit Tests	ProgramDataImportErrorHandler	The incoming data might not be correct and these tests will mock error bound incoming data	test_tbd.py
Task	1.1.3	Program Data Import UI			test_tbd.py
ST		Smoke Testing the UIs function	(1) Classes required in the import (2) UI	Smoke Testing is an initial reaction to the general size of the UI. Automation might be needed if we end up having a more involved UI - but this is not the case as of yet	test_tbd.py
FR	1.2	Constraint Data Import			test_tbd.py
UT		Constraint Data Import	Classes required in the import	These tests will mock the data source with python mocks	test_tbd.py
IT		Constraint Data Import + Connection to mirror data source	(1) incoming data stream (2) Constraint data importer	(1) will be a Dev mirror of the actual resource to be used in prod.	test_tbd.py
LT		Constraint Data Import + Connection to mirror data source	(1) incoming data stream (2) Constraint data importer	These test will mock a large amount of data coming in through (1) and test the results of (2)	test_tbd.py
Task	1.2.3	Constraint Data Import UI			test_tbd.py
ST		Smoke Testing the UIs function	(1) Classes required in the import (2) UI	Smoke Testing is an initial reaction to the general size of the UI. Automation might be needed if we end up having a more involved UI - but this is not the case as of yet	test_tbd.py
Feature	2	Data Export		*The Data Export component*	test_tbd.py
FR	2.1	Data Export Action			test_tbd.py
UT		Data Exporter Unit testing	Schedule Export Classes	Tests to assure output is accurately displayed and organized as desired	test_tbd.py
IT		Create output from a schedule	(1) Schedule class (2) Schedule Export class	After a schedule is created output should be reported to user and stored in proper format	test_tbd.py
ST		Smoke Testing Output	(1) UI tool (2) Data Exporter	Smoke testing file output. This can be done by comparing exported data to the schedule data	test_tbd.py
Task	2.1.1	Develop Associated UI			test_tbd.py
ST		Schedule displayed to user	(1) UI (2) Scheduler	Smoke test UI output. This can be done by comparing output to schedule data.	test_tbd.py
FR	2.1.1	Data Export to Format(s)			test_tbd.py
UT		Unit test all export formats	(1) Schedule export class	Output the proper format upon request	test_tbd.py
Task	2.1.1.2	Develop Schedule Consumer for Export(s)			test_tbd.py
IT		Schedule to export format	(1) Schedule (2) Schedule export class	Schedule must be converted to exportable state	test_tbd.py
FR	2.2	Final Data Visualization			test_tbd.py
ST		Exporting to specific format	Schedule export class	Reaction to data presentation in UI. This data must be presented to the user in organized and intuitive fashion.	test_tbd.py
ST		Smoke test UI schedule representation	UI tool	Reaction to data presentation in UI. This data must be presented to the user in organized and intuitive fashion.	test_tbd.py
Feature	3	Observation Data		*The Observation Data component*	test_tbd.py
FR	3.1	Data Identification			test_tbd.py
UT		Observation block unit testing	ObservationBlock class	The observation block will hold relevant data for observations and program identification.	test_tbd.py
IT		Observation blocks built with imported data	(1) ObservationBlock (2) Incoming data stream (3) program data importer	Integration of mock data transformed into an observation block	test_tbd.py
Task	3.1.1	Develop interal ID tracker for managed data.			test_tbd.py
LT		Unique identification of observation blocks	(1) ObservationBlock (2) incoming data stream (3) program data importer	Test the uniqueness of blocks with similar data	test_tbd.py
FR	3.2	Data Model			test_tbd.py
UT		proper transformation of incoming data into useful scheduling information	(1) data Transformer	Not all data from incoming source will be necessary	test_tbd.py
FR	3.2.1	Data View			test_tbd.py
ST		Review data to be used for scheduling	(1) ObservationBlock (2) UI tool	Smoke testing the ability for a user to reveiw data	test_tbd.py
Task	3.2.1.1	Develop Simplified UI View of TSO Data			test_tbd.py
UT		Schedule input parsing	(1) Observation block (2) UI Tool	User should be able to inquire about data being consumed by the scheduler,	test_tbd.py
FR	3.3	Data Persistence			test_tbd.py
UT		Rertrival of created schedules stored locally	(1) Data Importer (3) UI tool	After a schedule has been created, TSO can retrieve and display it if stored locally	test_tbd.py
ST		Past schedule accuracy	(1) Data Importer (3) UI tool	Smoke testing of an old schedule to assure state persistence	test_tbd.py
Task	3.3.1	Design strategy for persiting user data.			test_tbd.py
ST		User data persistance	(1) data exporting tool	***Waiting for solutioning/approval*** user information may be attached to schedules if needed	test_tbd.py
Feature	4	Scheduler		*The Scheduler component*	test_tbd.py
FR	4.1	Scheduling Selection			test_tbd.py
UT		Scheduler unit testing	Scheduler class	Test the creation of a schedule for various lengths of time	test_tbd.py
IT		Integrate data mirror with schedule	(1) Scheduler (2) Data Import classes	Tests the creation of a schedule which consumes mock data	test_tbd.py
LT		Stress Test Scheduling	Scheduler class	Tests the speed and function of the scheduler under heavy load	test_tbd.py
Task	4.1.2	Develop Schedule Structure			test_tbd.py
ST		Smoke testing a schedule with various structural attributes	(1) Scheduler (2) Data Export	Smoke testing will help explore schedules with various sizes of observation blocks used to build schedules	test_tbd.py
Task	4.1.3	Design Associated UI to select TSO Data			test_tbd.py
IT		Schedule changes via UI	(1) Scheduler (2) Data Import UI	These tests will assure that an optimal schedule is created with the insertion of user selected data	test_tbd.py
ST		Smoke testing UI overide	Scheduler	This test is used to explore user interaction with the scheduler	test_tbd.py
FR	4.2	Genetic Algorithm			test_tbd.py
UT		Unit test genetic algorithm	Genetic Algorithm class (ga.py)	Tests to mock the behaiviour of selecting the best observation	test_tbd.py
IT		Genetic Algorithm with cost training	(1) ga.py (2) Cost Hueristic class	Integrate cost function with the genetic algorithm class	test_tbd.py
QR	4.2.2	Optimize implementation to improve KPIs			test_tbd.py
PT		Cost accuracy and genetic performance	(1) Scheduler (2) Cost Function (3) ga.py (genetic algorithm)	Performance of scheduling will depend largely of the analysis of the cost of using a schedule. This will also depend on the genetic selection of observation blocks.	test_tbd.py
FR	4.2.1	Cost Function			test_tbd.py
UT		Cost Function unit testing	cost function	testing of the cost function with constraing mocking	test_tbd.py
IT		Cost Function with constraint interpretation	(1) cost function (2) constraint analyzer	Testing of the the cost function with constraint analyzer	test_tbd.py
Feature	5	Schedule Analysis		*The Schedule Analysis component*	test_tbd.py
Task	5.1	Investigate existing Schedules to determine optimization process			test_tbd.py
ST		Smoke testing exisitng schedule	(1) Schedule	Assures the selecting schedule is viable	test_tbd.py
Feature	6	Roles		*The Roles component*	test_tbd.py
FR	6.1	TSO User			test_tbd.py
ST		Schedule utilization	Entire System	A user should be able to integrate a schedule into their workflow	test_tbd.py
Task	6.1.1	Develop system executable delivery method			test_tbd.py
IT		Build and review Schedule	Entire System	TSO will be interacted with through a CLI	test_tbd.py
Feature	7	Performance		*The Performance component*	test_tbd.py
QR	7.1	Historical Performance			test_tbd.py
PT		Performance of scheduling using historic data	Entire System	Incoming data stream will be created using historic data. This data will consist of real celestial bodies/events that have been observable in the past. This will also include real constraints present at the time.	test_tbd.py
Task	7.1.1	Develop integration test of TSO given historical data			test_tbd.py
PT		Relative performance testing	Entire System	Create schedules with historic data and analyze performance relative to the existing CHFT scheduling system	test_tbd.py