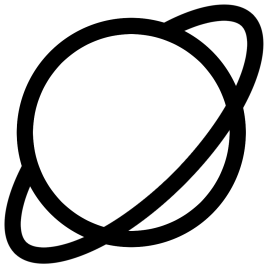


TSO

Progress Report 3

Sebastian Lopez
Gustavo Andres Murcia
Kirk Vander Ploeg
Elijah Ward

T | *Table of Contents*

T | Table of Contents **2**

1 | Project Description **3**
Summary 3
Problem Statement 4

2 | System Requirements **5**
Changes to Requirements 5

3 | Project Plan **6**
Changes to Project Plan 6

4 | System Design **7**
Technologies and Tools 7
Python 8
Conda 8
MySQL 8
Docker 8
User Interface 9
Traditional CLI Tool 9
Multi-Stage Application 9
Conceptual Context 10
Architecture Context 11
TSO Flow 11

5 | Implementation **12**
Workflow 12
TSO CLI 12
Core Modules 12

6 | Quality Assurance **13**
Quality Assurance Roadmap 13
QA Matrix 13

7 | Thinking Ahead **14**
Final Deliverable Roadmap 14

A | Appendix **15**

1 | *Project Description*

Summary

The following is the third of four reports on the progress of our chosen capstone project: ***Telescope Scheduling Optimization***. The project's initial goals are outlined in our first deliverable, and although some things have changed in our implementation of our solution 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. Team expectations have been outline in the *Team Outline* appendix item. This project is being supervised by Professor Pauline Barmby.

By working internally as a team, and leveraging our communications with Pauline, and with an MSE Database Specialist, we have been able to continue our implementation of the final TSO solution. Previous artifacts introduced in our first and second deliverables like the Functional Requirements, and various Project Planning items have been accordingly updated to reflect the current status towards the completion of the TSO system.

Progress Report 1	*Link to Resource*
Progress Report 2	*Link to Resource*
Team Outline	*Found in Appendix*

Problem Statement

The problem that TSO is addressing, remains identical to previous reports. With further analysis of the current CFHT data schema being done, more detailed questions have come up, but at a high-level our problem remains the same.

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*

2 | *System Requirements*

The following document is an update to the 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

Though little has changed at a high level in our Requirements, there have been changes during our implementation to the TSO system that reflect as minor scope changes in the requirements. The most important being our shift away from using the Genetic Algorithm process. This came from prebuilt functionalities coming from the Astroplan being identified as components that could improve the quality of the Schedule Selection process within TSO. So rather than implementing our own process from the ground up with the selection process, we will be adapting the use of Astroplan in our selection process with our own extension of its native selection process.

The only other change in scope is that of the User Interface. Since we have implemented the TSO CLI, we have seen how adaptable it is, to the already existing code that exists as part of other components (namely domain, and data importing), that implementing an external UI to view or manipulate data is not required. This also functions as a shift into TSO becoming a more native tool for the CFHT team as it works into their already existing process of scheduling and data handling.

System Requirements *Found in Appendix*

3 | *Project Plan*

We are continuously maintaining the various project planning documents utilized in past progress reports 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

Despite the lack of major functional changes to our project, we have incurred a large change to our plan due to some implementation details being change in our approach to the TSO solution. We have reconsidered our use of Genetic Algorithms as the methodology being used to rank and sort the various observations to be completed.

This change has reflected itself as a reduction in the time to implement the Scheduler, which has allowed some of the other items to be re-looked at in more detail for more accurate implementation.

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

4 | *System Design*

Although many aspects of the system design remain unchanged from our previous progress reports, certain low level components have changed through a natural evolution following several iterations of development. We initially took an abstract approach to thinking about what we were building because we knew that it was bound to change. Now that we have begun implementing the first version of the system, a clearer picture of how the final TSO system should behave is being envisioned by both our own team and our stakeholders. With this vision, we have been given the opportunity to reconsider our approach towards the design with respect to certain components of the system.

Since the delivery of our second progress report, we have made some calculated changes to the implementation of some of the high level components previously outlined. The overall flow of the system remains the same, but the method we use to optimize for the best possible schedule has been changed, causing very small changes to some design components. In particular, the team has decided to no longer use genetic algorithms and the **deap** library as part of the schedule optimization process. Instead, we will be leveraging more of the scheduling functionality built-in to **astroplan**. The library is highly extensible and provides APIs to create custom schedulers as well as custom constraints, in addition to a wide variety of each that come included.

Technologies and Tools

All of our core technology and tool decisions remain the same with the exception of the inclusion of using gRPC messaging to communicate with the CFHT infrastructure. This is because the team realized during this iteration that CFHT would like to move towards gRPC, but if TSO were to implement it right away, the tool would likely not be immediately useful. It is being suggested as a consideration for future teams who may continue this project.

In order to facilitate a consistent development environment amongst team members, we have implemented workflows using tools such as docker and conda. We use docker in order to deploy a local instance of MySQL DB with the MSE schema and test data scripts executed against it.

Python

Our use of Python continues and has solidified itself as our language of choice. We also use **pip** (the most popular python package management system) as a vital tool in delivering TSO to our stakeholders as a package in the pip registry. We also continue our plan to use **astropy** and **astroplan** within the system where astronomical computation is required. For the most part, the TSO command-line tool is leveraging much of the functionality included in these libraries and integrating it with the technology already in use by the CFHT team.

Conda

Conda is an open source package management and environment management system primarily intended for use with python. It has been a useful tool in the team's arsenal for the purpose of ensuring that with each code change, everyone has the necessary dependencies on their system in order to run it. This also helps to rule out any possibility of a faulty development environment if one encounters bugs during testing.

MySQL

Early in our process, the team received a MySQL DDL script from the CFHT DBA in order to help the team to understand the data model that our tool would need to integrate with. In order to deliver this integration, the team has used a MySQL docker image to spin up a local database to develop against. In addition, the team has made use of the **Flyway** tool in order to apply the schema. This combination of tools lends itself well to providing a seamless and platform-agnostic environment for all developers to collaborate on the data model. In addition, **Flyway** can be used to run subsequent migrations against the database once it is in a production environment and could be a useful tool for the CFHT team to adopt.

It is our current practice to transform data from the CFHT model into our own internal model as necessary, before transforming it again to interface with dependencies such as **astroplan**. This protects the project from becoming to coupled with either of the CFHT data model or the APIs of the libraries that we currently use.

Docker

The team has made extensive use of **Docker** on this project. The main benefit that it brings is a consistent environment for our database as we begin to run tests of the system against mock data. It is very helpful to keep all developers working under the same controlled environment, which allows all members to execute their tests in a consistent fashion.

User Interface

Although the decision to implement a CLI to expose TSO functionality to the user has been agreed upon, some low-level details remain for further clarification and development. A high-level understanding of how our users will interact with our system has been determined; a user will request a schedule for a specified date/time range to be scheduled. The placement of each **ObservingBlock** will then be assessed against all of the constraints, and the optimal schedule will be produced as output. At a lower level, our team has discussed two different methods that will dictate how users will ultimately use TSO. One method allows TSO to be used as a traditional functional CLI tool, and another exposes the system as a multi step process for the users.

Traditional CLI Tool

Our first option is to implement the CLI in a traditional functional way. This means that users will interact with the system through the invocation of a few powerful commands and as a result the desired schedule and feedback will be provided.

The benefit of this approach is that we could leverage the existing user's prior experience with traditional CLI tools. Users have fewer commands and varying parameters to remember as there will be a smaller set of commands to execute. This simpler approach requires less development time and allows for a simpler deployment of the final system. One downside is the amount of flexibility that TSO will offer, due to it's highly specialized functions.

Multi-Stage Application

Alternatively we have also discussed the idea of multi step interface that may grant the user with some added freedom and choice. This implementation will be multi-staged wherein each step of the of the process will rely on the user interacting with the system. As an example; A command for staging import data into the TSO context would proceed a command where a user will request feedback information on scheduling and another for requesting the schedule to be exported.

The key benefit we see from implementing this type of interaction is that the user will have influence over the proceedings and allowing them more flexibility over the workings of the system. Implementing functionality of lower specificity allows for future functionality to be added easily, but it also opens our team to higher development times.

Each approach will be presented to Pauline in order to receive her input on which would represent the CFHT's preferred way of interfacing with TSO.

Conceptual Context

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 approach is to divide each schedule into manageable units known as observation blocks. These Blocks are then grouped together to create a schedule which is executed and evaluated through the formulating of a **score** that evaluates the candidate schedule's optimization.

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**: Constraints whose evaluation can be computed without access to real-time data and can be computed offline. **(Deterministic)**
- **Dynamic**: Unpredictable conditions such as the weather give way to constraints that require real-time information gathering to assess. (**Non-Deterministic)**
- **Priority**: the observation of certain celestial bodies may be more important than others.

A schedule is then produced as a configuration of **ObservingBlocks**.

At the delivery of our second progress report, the team had sought to employ **genetic algorithms** as an optimization method in the backend of our system. Since then, the team has come together in discussion and decided to no longer follow this approach. It became increasingly evident that using a **genetic algorithm** in order to generate potentially thousands of candidate schedules and evaluate each one would be unnecessarily complex, redundant, and computationally expensive when compared to other methods of scheduling. The team has opted to employ a more traditional optimization method that works by iterating through each position in a schedule and inserting the Observation Block with the highest **score** for that position against a set of constraints. The team felt that the use of **genetic algorithms** was hindering progress due to a slight "reinventing of the wheel" in a scenario where that effort was not providing a benefit above more simple and proven alternative methods.

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

With the new design decisions, the flow has been slightly altered. The following reflects an updated version of how data will flow through the system and be used to create a schedule.

Firstly, proprietary information is consumed from the client's data model through the **DataImporter**. Following this, a collection of **ObservationRequest** 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 **astroplan.scheduling**, a module within the open source library **astroplan** that provides tools that assist with scheduling observation runs. The **Scheduler** module will first instantiate **astroplan ObservingBlock** objects from each of the **ObservationRequests** and their targets. Then, it will make use of the **ConstraintAggregator** object which will be used to iterate through all configured constraint names and initialize the **Constraint** objects using the **ConstraintBuilder**. All **Constraint** objects will conform to the interface supplied by the **astroplan** library. Some constraint objects are predefined within **astroplan** but others require up to date information in order to accurately compute the score of the constraint. Such objects are defined by TSO as **DynamicConstraint** objects. One such example of this is with the **WeatherConstraint** which requires a connection to an online **WeatherAPI** to retrieve the most up to date forecast for the time range at the site of the telescope. From here, the constrained **ObservingBlocks** can be supplied to an **astroplan Scheduler** object to assign a total *cost* (inversely, *fitness*) to the placement of a given **Observation Block** in a particular schedule timeslot.

Although it uses many of the same objects and design concepts as explained within *Progress Report 2*, our approach has changed slightly in that we are no longer using the **deap** library or genetic algorithms in order to optimize the schedule. The team has determined that leveraging more of the built-in functionality provided by **astroplan** will avoid "*reinventing the wheel*" and free up the team to create other features such as data export to a higher standard. In addition, the gain in schedule optimization using genetic algorithms is likely to be negligible when compared to a simpler, more efficient scheduling algorithm.

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

5 | *Implementation*

Since the delivery of Progress Report 2, our team has been both implementing new elements and refactoring original ones as the design evolves. Progress has been made in the development of each core module as well as to the CLI client application.

Workflow

The team has been diligent in following a good git workflow. The use of feature branches, pull requests and code reviews before master merges has been consistent. The team recognizes that other people may be working on this project in the future and is putting forth effort to ensure that the project is well documented and development environments are consistent so that it would be easy to onboard new contributors. Our use of descriptive PRs, sound branching patterns and flexible package managers such as conda and pip.

TSO CLI

Since Progress Report 2, the structure of the CLI has undergone several iterations in order to follow an extensible plug-and-play command structure. This allows for features and commands to easily be added to the CLI tool if more core functionality were to be added in the future. Descriptive usage messages allow the user to easily intuit the flow of the tool. In the future the team plans to put together a simple manual to package with the system for an improved user experience.

Core Modules

Implementation of all fundamental components of TSO are well underway. The following TSO Entities now have implementations and their development continues; **Constraint**, **Dynamic Constraint**, **Constraint Aggregator**, **Observation Request**, **Scheduler**, **Transitioner**, **DataImporter**. Many core modules have recently undergone some slight changes due to the change in our approach to implementing the **Scheduler**. Although these changes may have set us back slightly in the short term, they will bolster the success of the team as we move toward the final delivery.

In most cases, modules that either are a **noun** or end in a **noun** are implemented as a **class**, where modules that either are a **verb** or end in a **verb** are implemented as a **function**. Wherever possible, the team has been creating unit tests for code as it is implemented.

TSO GitHub Repository *Link to Resource*

6 | *Quality Assurance*

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

Quality Assurance (QA) efforts continue through the maintenance of two front facing documents that reflect the QA work. The QA matrix has been updated to reflect recent changes to the scheduler module but largely remains the same as changes are limited to features which involved the genetic algorithm. Tests will continue to be implemented using the ***pytest*** framework. This will allow each member of our team to contribute in a seamless way towards their respective QA tasks. Upon full implementation of these tasks, with the use of **pytest-html**, we will be able to produce an easy to access Test Report Document that both the client and members of our team will have access to.

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.

Each of these QA tasks represent a *Test Case* we will be testing as part of the final testing for TSO. Line items in the QA Matrix with a ***Source*** (ie: the file that holds those tests), are ones that have been implemented thus far.

One addition to the QA Matrix now that we have commenced our testing efforts, is the addition of a *Status* column where each test case will be said to be either; Failing, Passing, or still being written (in progress).

QA Matrix *Found in Appendix*

Final Deliverable Roadmap

As we now enter the final stage of this project we have a clear view of what remains to be done in order to reach our initial project goals and deliver a system that is compliant with stakeholder needs. Since our last deliverable we have focused our combined efforts solely on implementing the major modules of our system. Throughout our development process we have outlined some flaws in prior designs of what we thought TSO would be.

We have recognized that using a genetic algorithm to optimize schedules is overly complex for our use case of a selection methodology. Alternatives that are less computationally intense will lead to equally satisfying results. We have also identified our prior thought of using Protocol Buffers as a communication protocol is not necessary. However, we do see it as a potential for future iterations of TSO but at the moment there is no need for it as it is not compliant with the CHFT technologies.

Although these flaws have presented us with additional work in the design process, there are two points that can be gathered from this. First is the timing of the discovery of these flaws. Since they manifested themselves during development, early architectural design decisions have allowed for these changes to be decoupled from other components in the project. This will provide us with the opportunity to continue on with the development seamlessly. In turn, these flaws in our prior thinking have actually caused our critical path to have a reduced development time. The other point is that we now have a very clear understanding of what is required for us to move forward with the implementation of TSO. This will ensure that we create a system that fulfills all requirements and is delivered on time.

With regards to our next steps, we are currently developing the final details of the data importing and scheduler entities. We need to be certain that all the needed data is properly adapted to our own schema so it can be properly digested by the scheduler. With these up and running we can now focus our attention on the remaining entities that will dictate how the created schedules will be made available to the user, in addition to working on the CLI. These responsibilities include the following:

- **Data Export**: Translate scheduling output to user consumable format
- **Data Review**: Allow a user to review schedule prior to being output
- **Feedback Reports**: Output why observations were chosen and why others were not
- **Historical Performance**: Provided with historical observation data, schedule an appropriate observation session.

A | *Appendix*

Team Outline

Context Diagram

Architectural Diagram

Entity Outline

System Requirements

Project Tasks

Project Timeline

Project Work Breakdown Structure

QA Matrix

Team Outline

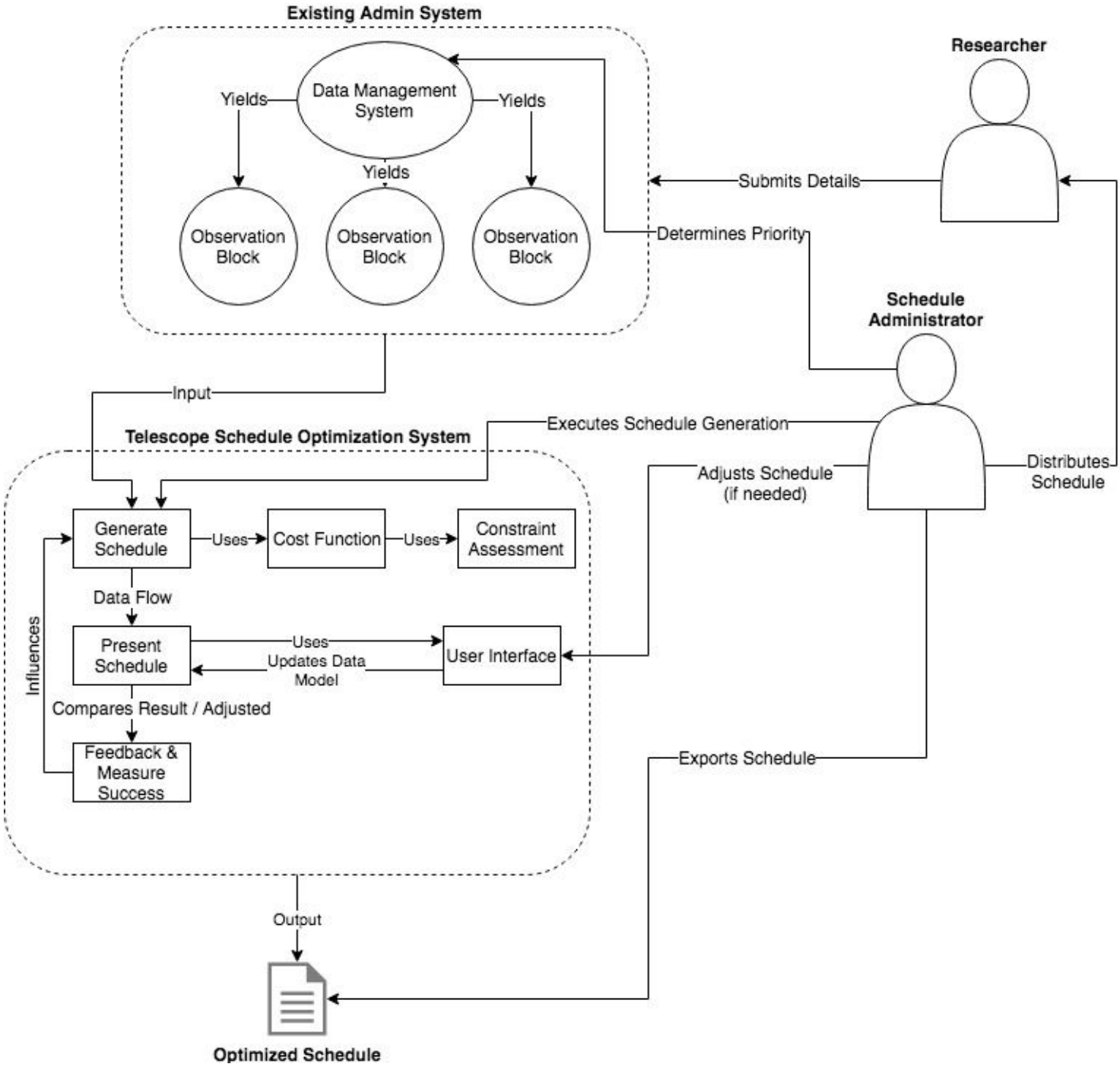
Team Agreements

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.

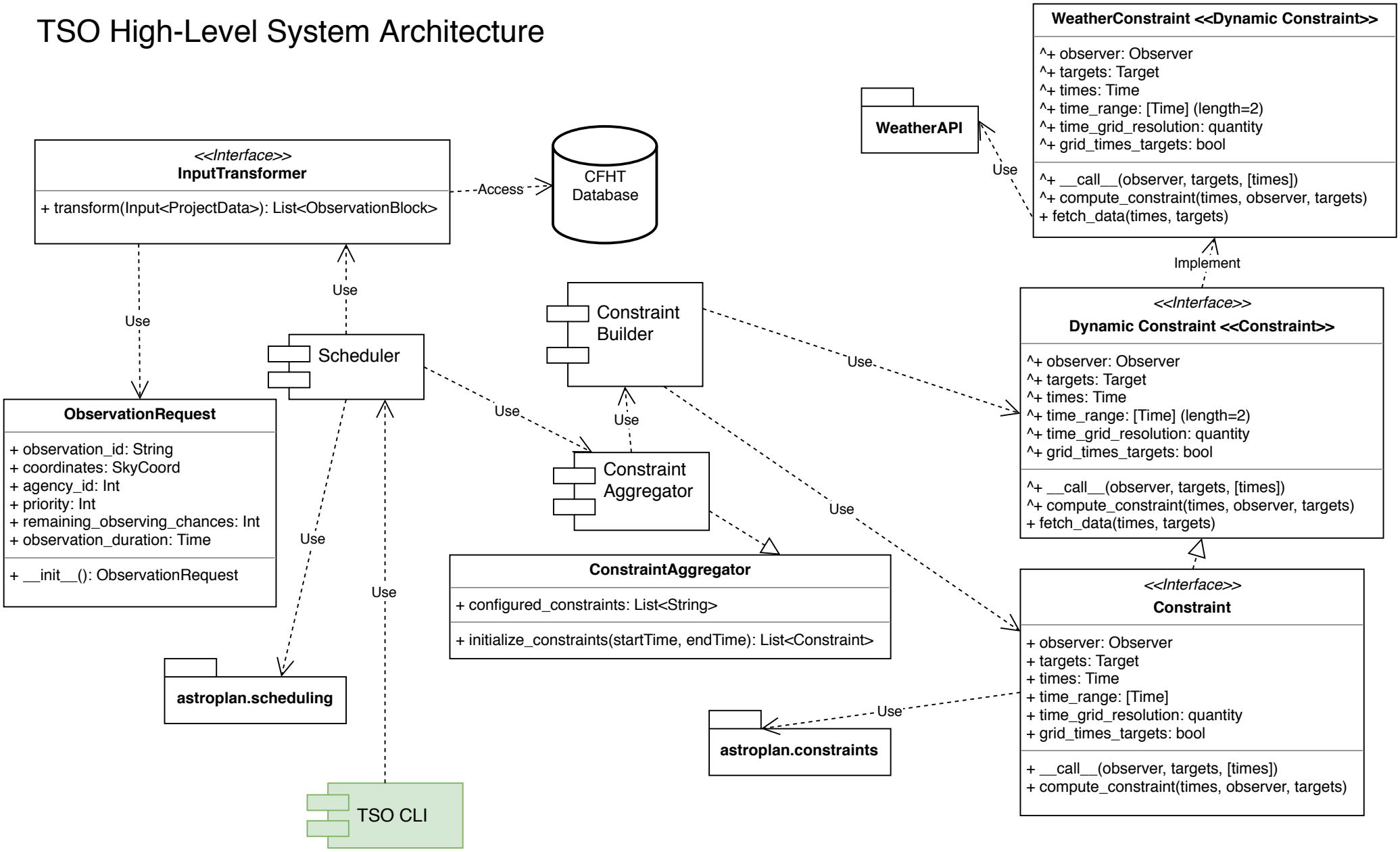
Role Assignment Table

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



TSO High-Level System Architecture



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 Request	The **ObservationRequest** 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. Formerly this was referred to in the TSO domain as an **ObservingBlock**. This caused confusion due to a conflict with a directly related entity in the **astroplan** domain, so the name was modified.
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. This will be where many of the other entities are instantiated and consumed.
Observing Block	The **ObservingBlock** is an **astroplan** entity that observation events must be converted into in order to interface with **astroplan** constraint and scheduler objects.
Constraint Aggregator	Used by the **Scheduler** at initialization, the **ConstraintAggregator** is a module that iterates through the configured **Constraints** and initializes them using the **ConstraintBuilder** prior to evaluation against **ObservingBlocks**.
Constraint Builder	The **ConstraintBuilder** initializes the set of logic which can take the context surrounding the configuration and placement of an **ObservingBlock** and apply 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 **astroplan**, to solve for constraints such as the location of the target in the sky, as well as other resources such as a **Weather API** to determine dynamic conditions such as visibility. Each constraint will be initialized once per scheduling session, after which **ObservationBlocks** can be evaluated against them.
Constraint	Constraints exist in two basic variations, dynamic and fixed. The default **Constraint** 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 **Constraint**.
Dynamic Constraint	Inheriting from the Constraint interface, this object provides a more specified version of Constraint that requires a step of accessing data from some frequently updated source before it can be assessed. This is because these are constraints that change frequently and are more likely to change between schedule generation and the scheduled observation itself, making up to date information a necessity.

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 model	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.		***From CFHT Notes*** *Input and output formats are variable (xml, JSON, csv, etc) - current system does not have its own visualization tools so users export data into a format that works with their favorite tools.*
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		Since we are taking a CLI centered approach this will be a simplified view. Depending on what CLI path we take it may be available automatically to the user. Or the user may have to explicitly specify the data they want to vie
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	Revoked	Data	The TSO system shall persist data between sessions of use	This requirement is no longer needed as the system will not accessed through a regstered users and a respective authentication procedure. So user data will no longer need to be persistent	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**	Selection Algorithm	Review	Scheduling	The TSO will use a selection process with astroplan which uses 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		
QR	**4.2.2**	Selection Algorithm Performance	Review	Quality	TSO's use of the selection algorithm will be tuned to meet speed and quality KPIs set by stakeholders	If the selection algorithm is tunable, it shall be tunable to paramters that are concrete to the 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		
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		D
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			D
Task	1.1.2	Develop Program Data Input Transformer	Dev	Gustavo			IP
QR	1.1.2	Error Checking Incoming Data	QA	Kirk			IP
Task	1.1.3	Develop Associated UI	UI	Kirk			IP
FR	1.2	Constraint Data Import		ALL			D
Task	1.2.1	Design Constraint Data Import	Design	ALL			D
Task	1.2.2	Develop Constraint Data Import	Dev	Sebastian			IP
Task	1.2.3	Develop Associated UI	UI	Gustavo			IP
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			IP
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			D
Task	3.2.3	Develop TSO Data Model	Dev	Elijah			IP
FR	3.2.1	Data View		ALL			IP
Task	3.2.1.1	Develop Simplified UI View of TSO Data	UI	Kirk			IP
FR	3.3	Data Persistence		ALL			IP
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	Selection Algorithm		ALL		Scope Reduction	IP
Task	4.2.1	Design Selection Algorithm	Design	ALL		Scope Reduction	IP
Task	4.2.2	Develop Selection Algorithm	Dev	Elijah		Scope Reduction	W
QR	4.2.2	Optimize implementation to improve KPIs	Optimize	Gustavo			W
FR	4.2.1	Cost Function		ALL		Scope Reduction	W
Task	4.2.1.1	Design Cost Function	Design	ALL		Scope Reduction	W
Task	4.2.1.2	Develop Cost Function	Dev	Sebastian		Scope Reduction	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

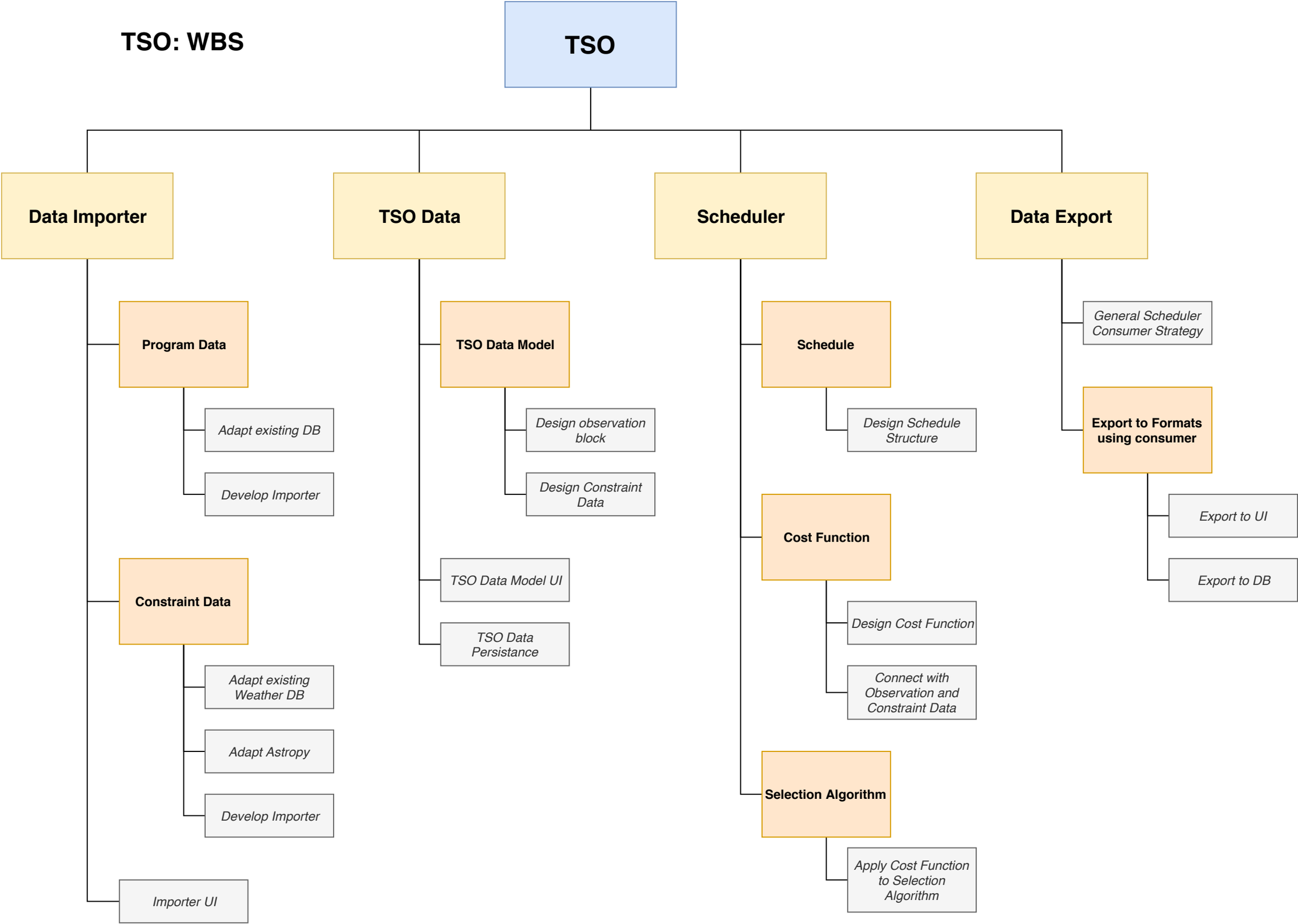
(Column = Week as of Monday)

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		D																						;						
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			D								;	;																		
Task	1.1.2	Develop Program Data Input Transformer	Dev	Gustavo			IP																	-	-	-								
QR	1.1.2	Error Checking Incoming Data	QA	Kirk			IP																		-	-								
Task	1.1.3	Develop Associated UI	UI	Kirk			IP																			-								
FR	1.2	Constraint Data Import		ALL			D								;	;	;																	
Task	1.2.1	Design Constraint Data Import	Design	ALL			D								;	;	;																	
Task	1.2.2	Develop Constraint Data Import	Dev	Sebastian			IP																		-	-	-							
Task	1.2.3	Develop Associated UI	UI	Gustavo			IP																				-							
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			IP																						-					
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			D										;	;	;															
Task	3.2.3	Develop TSO Data Model	Dev	Elijah			IP																		-	-	-							
FR	3.2.1	Data View		ALL			IP																					-						
Task	3.2.1.1	Develop Simplified UI View of TSO Data	UI	Kirk			IP																					-						
FR	3.3	Data Persistence		ALL			IP																						-	-				
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	Selection Algorithm		ALL		Scope Reduction	IP								;	;												-	-	-				
Task	4.2.1	Design Selection Algorithm	Design	ALL		Scope Reduction	IP								;	;												-						
Task	4.2.2	Develop Selection Algorithm	Dev	Elijah		Scope Reduction	W													;	;	;	;					-	-					
QR	4.2.2	Optimize implementation to improve KPIs	Optimize	Gustavo			W																					.	.	.				
FR	4.2.1	Cost Function		ALL		Scope Reduction	W																						.	.	.			
Task	4.2.1.1	Design Cost Function	Design	ALL		Scope Reduction	W																						.	.				
Task	4.2.1.2	Develop Cost Function	Dev	Sebastian		Scope Reduction	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
Selection Algorithm
Apply Cost Function to Selection Algorithm
General Scheduler Consumer Strategy
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

Status Legend	
Passing	.
Failing	;
In Progress	-

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	Status
Feature	1	Data Import		*The Data Import component*		-
FR	1.1	Program Data Import				;
UT		Program Data Importer Unit Testing	Classes required in the import	These tests will mock the data source with python mocks	test_importer.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_importer.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_importer.py	-
QR	1.1.2	Error Checking Incoming Data				;
UT		ProgramDataImportErrorHandler Unit Tests	ProgramDataImportErrorHandler	The incoming data might not be correct and these tests will mock error bound incoming data	test_importer.py	;
Task	1.1.3	Program Data Import UI				.
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	Manual Test	.
FR	1.2	Constraint Data Import				-
UT		Constraint Data Import	Classes required in the import	These tests will mock the data source with python mocks	test_importer.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_importer.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_scheduler.py	-
Task	1.2.3	Constraint Data Import UI				-
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	Manual Test	-
Feature	2	Data Export		*The Data Export component*		-
FR	2.1	Data Export Action				-
UT		Data Exporter Unit testing	Schedule Export Classes	Tests to assure output is accurately displayed and organized as desired	Manual Test	-
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		-
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	Manual Test	-
Task	2.1.1	Develop Associated UI				-
ST		Schedule displayed to user	(1) UI (2) Scheduler	Smoke test UI output. This can be done by comparing output to schedule data.	Manual Test	-
FR	2.1.1	Data Export to Format(s)				-
UT		Unit test all export formats	(1) Schedule export class	Output the proper format upon request		-
Task	2.1.1.2	Develop Schedule Consumer for Export(s)				-
IT		Schedule to export format	(1) Schedule (2) Schedule export class	Schedule must be converted to exportable state		-
FR	2.2	Final Data Visualization				-
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.	Manual Test	-
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.	Manual Test	-
Feature	3	Observation Data		*The Observation Data component*		-
FR	3.1	Data Identification			test_observation.py	-
UT		Observation block unit testing	ObservationBlock class	The observation block will hold relevant data for observations and program identification.	test_observation.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_scheduler.py	;
ST		Observation block creation according to mock data	(1) ObservationBlock (2) Incoming data stream (3) program data importer	Use mock data to import random constraints, targets, and observation requests.	Manual Test	.
Task	3.1.1	Develop interal ID tracker for managed data.			test_observation.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_observation.py	-
FR	3.2	Data Model			test_observation.py	-
UT		proper transformation of incoming data into useful scheduling information	(1) data Transformer	Not all data from incoming source will be necessary	test_observation.py	-
FR	3.2.1	Data View				-
ST		Review data to be used for scheduling	(1) ObservationBlock (2) UI tool	Smoke testing the ability for a user to reveiw data	Manual Test	-
Task	3.2.1.1	Develop Simplified UI View of TSO Data				-
UT		Schedule input parsing	(1) Observation block (2) UI Tool	User should be able to inquire about data being consumed by the scheduler,		-
FR	3.3	Data Persistence				-
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		-
ST		Past schedule accuracy	(1) Data Importer (3) UI tool	Smoke testing of an old schedule to assure state persistence	Manual Test	-
Task	3.3.1	Design strategy for persiting user data.				-
ST		User data persistance	(1) data exporting tool	***Waiting for solutioning/approval*** user information may be attached to schedules if needed	Manual Test	-
Feature	4	Scheduler		*The Scheduler component*		-
FR	4.1	Scheduling Selection			test_scheduler.py	;
UT		Scheduler unit testing	Scheduler class	Test the creation of a schedule for various lengths of time	test_scheduler.py	;
IT		Integrate data mirror with schedule	(1) Scheduler (2) Data Import classes	Tests the creation of a schedule which consumes mock data	test_scheduler.py	;
LT		Stress Test Scheduling	Scheduler class	Tests the speed and function of the scheduler under heavy load	test_scheduler.py	;
Task	4.1.2	Develop Schedule Structure			test_scheduler.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	Manual Test	.
Task	4.1.3	Design Associated UI to select TSO Data				-
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		-
ST		Smoke testing UI overide	Scheduler	This test is used to explore user interaction with the scheduler	Manual Test	.
FR	4.2	Selection Algorithm			test_scheduler.py	-
UT		unit test scheduler class	(1) scheduler class	Tests to mock the behaiviour of selecting the best observation	test_scheduler.py	;
IT		test scheduler class with constrain and optimizer	(1) scheduler class (2) Cost Hueristic class (3) optimize_schedule class	Integrate cost function and optimization with scheduler	test_scheduler.py	;
QR	4.2.2	Optimize implementation to improve KPIs			test_scheduler.py	-
PT		Cost accuracy and schedule scoring performance	(1) optimize_shedule class (2) Cost Function (3) scheduler class	Performance of scheduling will depend largely of the analysis of the cost of using a schedule. This will also depend on the scheduler selection of observation blocks.	test_scheduler.py	-
FR	4.2.1	Cost Function			test_scheduler.py	-
UT		Cost Function unit testing	cost function	testing of the cost function with constraing mocking	test_scheduler.py	-
IT		Cost Function with constraint interpretation	(1) cost function (2) constraint analyzer	Testing of the the cost function with constraint analyzer	test_scheduler.py	-
Feature	5	Schedule Analysis		*The Schedule Analysis component*		-
Task	5.1	Investigate existing Schedules to determine optimization process				-
ST		Smoke testing exisitng schedule	(1) Schedule	Assures the selecting schedule is viable	Manual Test	-
Feature	6	Roles		*The Roles component*		-
FR	6.1	TSO User				-
ST		Schedule utilization	Entire System	A user should be able to integrate a schedule into their workflow with ease. Smoke testing this scenario is necessary to fulfill a users needs.	Manual Test	-
Task	6.1.1	Develop system executable delivery method				-
IT		Build and review Schedule	Entire System	TSO will be interacted with through a CLI		-
Feature	7	Performance		*The Performance component*		-
QR	7.1	Historical Performance			demo_integration_test.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.	demo_integration_test.py	-
Task	7.1.1	Develop integration test of TSO given historical data			demo_integration_test.py	-
PT		Relative performance testing	Entire System	Create schedules with historic data and analyze performance relative to the existing CHFT scheduling system	demo_integration_test.py	-