Members

Aditya Nair, Jayden Cooke, Jun Ho Jin, Nathan Situ, Priyankit Singh

Project 1 plan

Group 6

Contents

[Work Breakdown Structure 2](#_Toc457830820)

[List of Tasks 2](#_Toc457830821)

[1.0 Analysis and Design 2](#_Toc457830822)

[2.0 Code 3](#_Toc457830823)

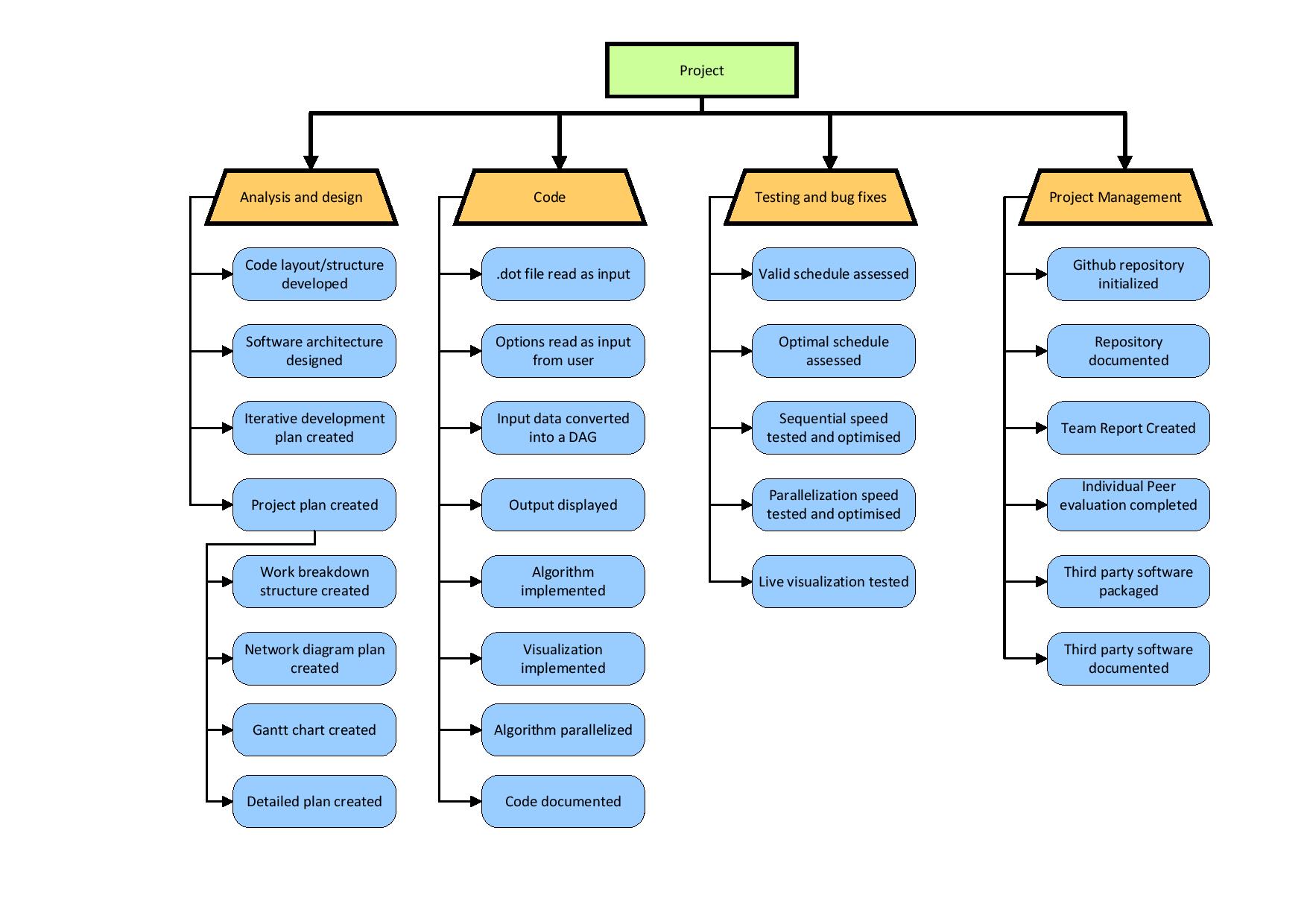
[3.0 Testing and Bug Fixes 4](#_Toc457830824)

[4.0 Project Management 4](#_Toc457830825)

[Project Network 6](#_Toc457830826)

[Gantt Chart 7](#_Toc457830827)

# Work Breakdown Structure



# List of Tasks

### Analysis and Design

#### 1.1 Code layout/structure developed

Coding conventions such as naming variables and classes etc. This is to allow more understandability when needing to review the code making the process faster. Classes must be implemented according to the layout specified and organized in a way that is continuous throughout the entire project.

* Time: Should be constant
* Resources: Everyone
* Dependencies: Done after Project planning (1.3)

#### 1.2 Software architecture designed

Design program architecture so that we can understand the whole structure of the software and build on it so that the code structure will remain clean.

* Time: 2 days of planning
* Resources: Everyone should be planning it so that everyone has an understanding of the whole structure
* Dependencies: Done after Project planning (1.3)

#### 1.3 Project plan created

* Work breakdown structure (WBS) created
* Networking diagram plan created
* Gantt chart created
* Detailed plan created
* Planning the project helps us so that it reduces scope creep and allow us to finish the project in a timely manner. It allows to organize time and work in such a way that certain aspects of the project can be implemented at certain time ensuring a continuous progression rather than burstly additions.
* Time: 3 days of planning
* Resources: Once the tasks has been created 1 person can work on the WBS, another on the diagram diagram plan, another on the Gantt chart and the rest on the detailed plan.
* Dependencies: Start of project

#### 1.4 Iterative development plan created

Ideally the foundations of the project will be complete by a week meaning that the software will display an output. However, it does not need to be optimal at this stage. Iterations on the project will be done each week so that we can clean and improve on the existing code to obtain an optimal schedule further on. This involves fixing errors, certain classes, and methods to improve performance. (Trello)

* Time: 1 Iteration should be equivalent to 1 week
* Resources: Everyone since it is the whole project.
* Dependencies: Design software architecture

### 2.0 Code

#### 2.1 .dot file read as input

This part is important to be correct since if it’s wrong it will propagate through the whole code. This, as with much of the I/O, needs to be done through interfaces, so that other parts of the project can be developed in parallel. The software will be reading a .dot file.

* Time: 4hr
* Resources: 1 person will work on it since it is a small task
* Dependencies: To be done after designing software architecture (1.2).

#### 2.2 Options read as input from user

Adjusting the main method to allow various parameters to be passed in for things such as the visualization interface, or the choice of parallel or serial running. Additionally, it must be fairly easy to add further parameters in future.

* Time: 2hr
* Resources: 1 person will work on it since it is a small task
* Dependencies: To be done after designing software architecture (1.2).

#### 2.3 Input data converted into a DAG

When the input is read, the data needs to be transformed into a usable format that can be represented as a DAG.

Time: 1 day

Resources: 1 person

Dependencies: Cannot be done before 2.1 and 2.2

#### 2.4 Output displayed

Making sure that the output is in the same layout as described in the project brief. Creating a .dot file from the digraph structure, as fast as possible.

* Time: < 1hr once the algorithm is done all we need to do is display it
* Resources: 1 person will work on it since it is a small task
* Dependencies: To be done after 2.3

#### 2.5 Algorithm implemented

The algorithm for traversing the graph and finding the optimal time to sort it. (2 parts, potentially more) The first part, for the initial milestone submission involves researching the algorithm, deciding which of branch and bound or A\* to implement, and implementing it to get a valid solution. The second part involves optimising the speed of the algorithm, by any means possible, while still returning an optimal solution.

* Time: 1 week
* Resources: At least 2 people should work on this part (pair programming) since it is the core part of deciding whether the program functions properly or not. There will be an observer and navigator where the observer will be constantly reviewing the code as the navigator types it down to make sure that it is correct.
* Dependencies: 2.3

#### 2.6 Visualization implemented

Visualization of the code working to allow our client to see how the algorithm is playing in the background. Either using external graph visualization libraries or standard GUI libraries. The key difficulty here is figuring out how to show the information without overloading the user or slowing the runtime that much.

* Time: 3 days
* Resources: 1 or 2 people.
* Dependencies: Need to implement algorithm in parallel (2.7) before visualization.

#### 2.7 Algorithm parallelized

Initially the first completed part of the project would be run sequentially. Using more than one processor (parallelization) would allow the program to run faster. This task would involve the use of parallelization libraries, and allowing the parallelization to be toggled with a command line parameter.

* Time: 1 week
* Resources: 2 people
* Dependencies: Need implementation (2.5) before parallelization.

#### 2.8 Code documented

Documenting the code on the whole software architecture. Ensuring that all team members follow the same conventions. Any complex sections should be commented, and larger, simpler sections have comments illustrating what they do. All classes and methods need to have formal Javadoc created.

* Time: At most 2 days.
* Resources: flexible this can be done at the end which means everyone can do it. At Least one person is needed for this.
* Dependencies: Documentation must be done at the end of coding phase.

### Testing and Bug Fixes

#### 3.1 Valid Schedule assessed

Verifying that the schedule does in fact yield the correct solution on a wide range of input, including normal, extreme and abnormal input. This will be done using JUnit test cases. The answer does not need to be the optimal solution, but has to be a valid solution. The test cases made during this step will likely be the skeleton for subsequent test cases, so may take longer.

* Time: 1 day to write test cases.
* Resources: 1 person does this task.
* Dependencies: Schedule assessment will be done after coding (2.x). It Will be done in parallel with 3.2.

#### 3.2 Optimal schedule assessed

Similar to the valid schedule assessment, but instead testing that the output is the optimal solution for the given input. This means that the path created must be optimized by some algorithm, and will be tested using JUnit test cases.

* Time: 1 day to write test cases.
* Resources: 1 person does this task.
* Dependencies: Schedule assessment will be done after coding (2.x). Will be done in parallel with 3.1.

#### 3.3 Sequential speed tested and optimized

Testing the speed of the program while running on a single processor. Requires adding a time function to existing tests.

* Time: 1 day
* Resources:1 person to write test cases.
* Dependencies: Speed testing will be carried out after schedule assessment (3.1 and 3.2).

#### 3.4 Parallelization speed tested and optimized

Testing the speed while running on multiple processors. Can be tested using existing tests, provided that all data structures are thread safe.

* Time: 1 day
* Resources: 1 person to write test cases.
* Dependencies: Speed testing will be carried out after schedule assessment (3.1 and 3.2).

#### 3.5 Live visualization tested

A visual test to see if the interface is showing the correct graph with the various measurements about the searching and nodes. This is done in order to check if the visual function is working as intended and that the optimal solution is also the one being displayed.

* Time: 3 hours.
* Resources: 1 person.
* Dependencies: This has to be done after implementing visualization (2.6).

### 4.0 Project Management

#### 4.1 GitHub repository initialized

Creating initial files, such as the gitignore and the initial commit with a main method. Adding all key people as contributors to the repository.

* Time: 30min
* Resources: 1 person (The repository creator)
* Dependencies: None

#### 4.2 Repository documented

Creating and setting up the GitHub documentation features, such as the wiki, the issues and the milestones. Establishing conventions for its use, and ensuring all members are aware of them.

* Time: 3 days
* Resources: 1 person
* Dependencies: Must be done after GitHub repository is created (4.1)

#### 4.3 Team Report Created

Overall assessment of group project, commenting on methods and tool used, problems encountered, and development lifecycles. Detailing any aspects of the project that aren’t obvious by the final product.

* Time: 7 days
* Resources: Everyone
* Dependencies: Can work on throughout project, but the final demo should be finished to complete the report.

#### 4.4 Individual Peer evaluation completed

Filling out a form about the amount of work undertaken by each member to ensure team members who did less than their share are penalized.

* Time: 2 hours
* Resources: Everyone
* Dependencies: Can only be done after all work is handed in (end of project)

#### 4.5 Third party software packaged

This step includes ensuring that all external libraries are packaged into the .jar file that we submit, so no installation has to be done by the client. This will likely involve the use of a build tool.

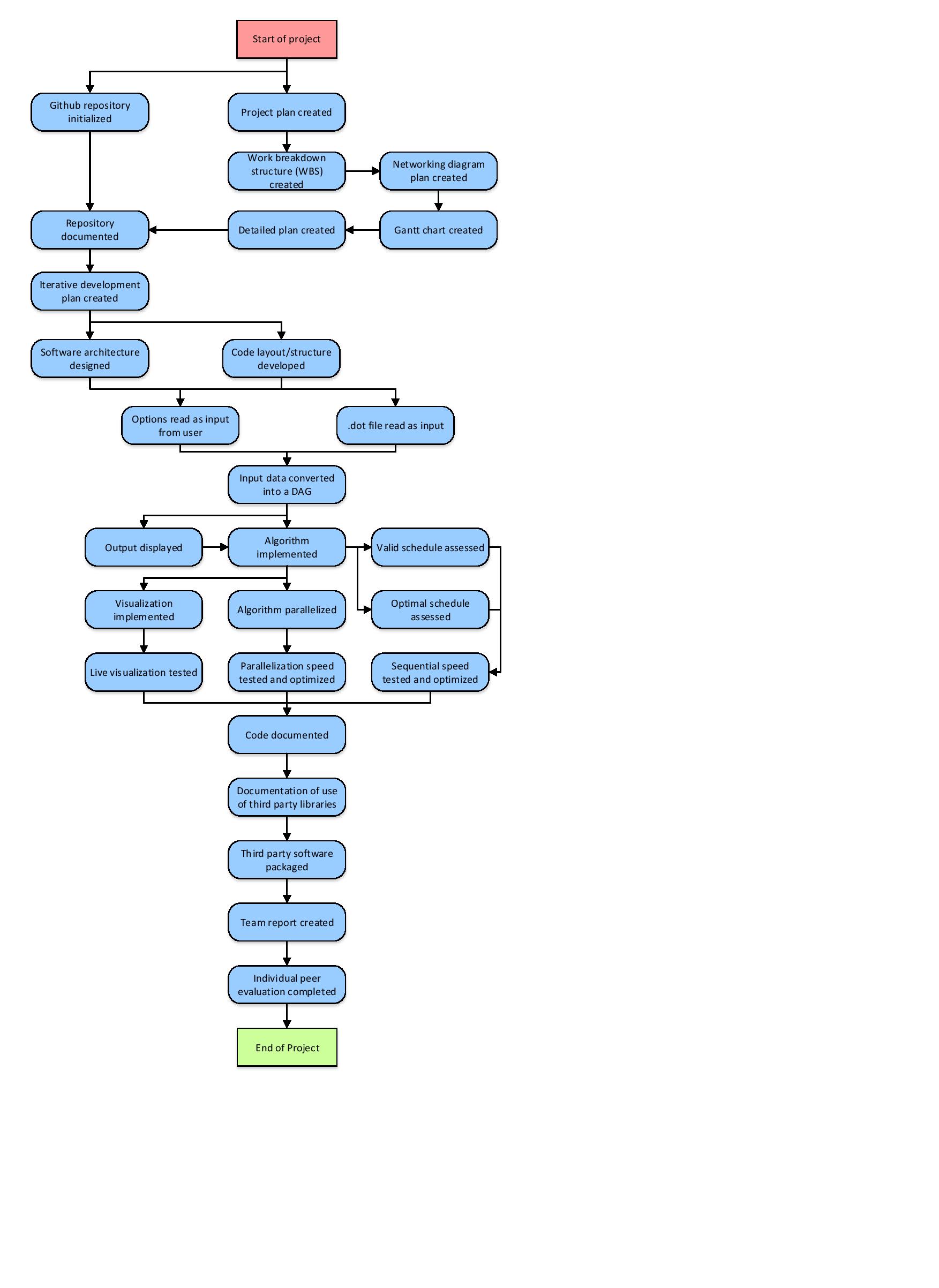
* Time: 4 hours
* Resources: 1 person
* Dependencies: To work on when all of the required third party software is added into the program.

#### 4.6 Third party software documented

The process of ensuring that all third party libraries are documented and kept separate from code that we wrote. Where possible, this will be done through changes in the directory structure of the project.

* Time: 3 hours
* Resources: At least one person, but ideally the whole team will review all code.
* Dependencies: To be completed alongside third-party implementation (4.5)

# Project Network



# Gantt Chart