



PROJECT 1

PLAN DOCUMENT

Group 6

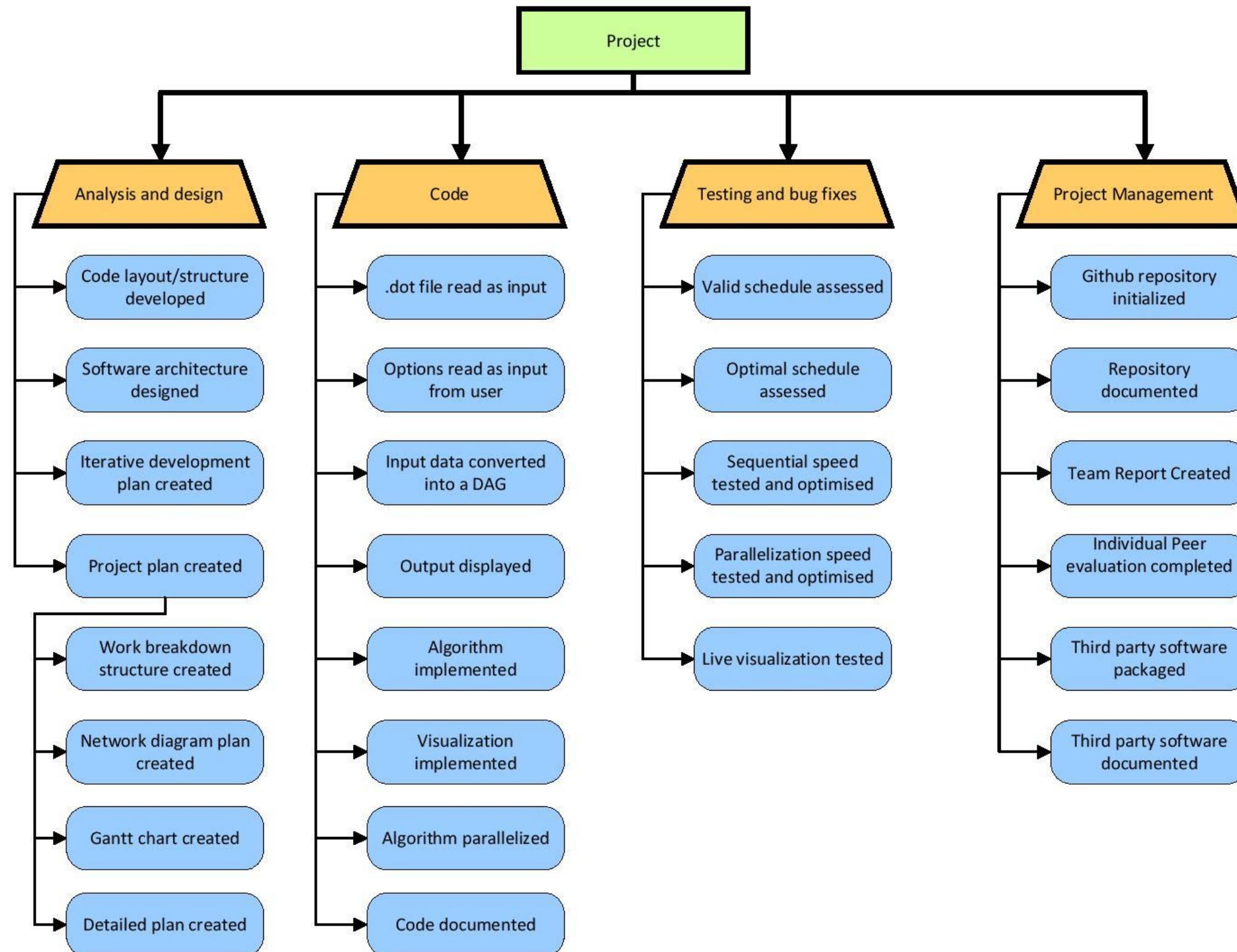
[Members](#)

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

Contents

Work Breakdown Structure	1
List of Tasks	2
1 Analysis and Design.....	2
1.1 Code layout/structure developed.....	2
1.2 Software architecture designed.....	2
1.3 Project plan created (10 days)	2
1.4 Iterative development plan created	2
2 Code	3
2.1 .dot file read as input.....	3
2.2 Options read as input from user (Final Milestone).....	3
2.3 Input data converted into a DAG	3
2.4 Output displayed.....	3
2.5 Algorithm implemented (First and Final Milestone).....	3
2.6 Visualization implemented (Final Milestone)	4
2.7 Algorithm parallelized (Final Milestone).....	4
2.8 Code documented (First Milestone and Final Milestone)	4
3 Testing and Bug Fixes.....	4
3.1 Valid Schedule assessed (First milestone)	4
3.2 Optimal schedule assessed (Final Milestone).....	4
3.3 Sequential speed tested and optimized (Final Milestone)	5
3.4 Parallelization speed tested and optimized (Final Milestone).....	5
3.5 Live visualization tested	5
4 Project Management	5
4.1 GitHub repository initialized.....	5
4.2 Repository documented.....	5
4.3 Team report created	5
4.4 Individual peer evaluation completed	6
4.5 Third party software packaged	6
4.6 Third party software documented	6
Project Network.....	7
Critical path.....	8
Project Network critical path	8
Longest path assessment.....	9
Working.....	9
Gantt Chart	10

Work Breakdown Structure



List of Tasks

1 Analysis and Design

1.1 Code layout/structure developed

Coding conventions such as naming variables and classes etc. This is to allow 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: 1 days

Resources: 2 people would be doing this part in parallel to Software architecture design

Dependencies: Done after Iterative development plan created (1.4). It will be done in parallel with software architecture designed (1.2).

1.2 Software architecture designed

Creating a design program architecture is essential to understand the whole structure of the software and build on it so that the code structure will remain clean. These are demonstrated through the use of UML diagrams.

Time: 2 days of planning

Resources: We will be using UML software and paper to create the diagrams. 3 people will be doing this task in parallel with the code layout development

Dependencies: Done after Iterative development plan created (1.4). It will be done in parallel with code layout/structure development (1.2).

1.3 Project plan created (10 days)

- Work breakdown structure (WBS) created – 1 day
- Networking diagram plan created – 2 day
- Gantt chart created – 2 day
- Detailed plan created – 1 day

Planning the project helps us so that it reduces scope creep and allow us to finish the project in a timely manner. It allows us 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.

Time: 10 days of planning

Resources: Once the tasks have been created 1 person can work on the WBS, another on the 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 are 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.

Time: 1 day

Resources: 2 people will be working on this in parallel to repository documentary. We will be using Trello as a mean to organize our tasks for agile development

Dependencies: Done after GitHub repository documented (4.2)

2 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: 4 hours

Resources: 1 person will work on it since it is a small task

Dependencies: To be done after designing software architecture (1.2) and code layout/structure developed (1.1). In parallel with options read as input from user.

2.2 Options read as input from user (Final Milestone)

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: 2 hours

Resources: 1 person will work on it since it is a small task

Dependencies: To be done after designing software architecture (1.2) and code layout/structure developed (1.1). In parallel with .dot file read as input.

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: 5 hours

Resources: 1 person

Dependencies: To be done after .dot file read as input (2.1) and options read as input from user (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: 2 hours. 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 the input data is converted into a DAG (2.3). In parallel with algorithm implemented.

2.5 Algorithm implemented (First and Final Milestone)

The algorithm for traversing the graph and finding the optimal time to sort it, this is essentially 2 parts. The first part, for the first milestone submission involves researching the algorithm, deciding whether to use branch and bound or A* to implement. Then getting a valid schedule for the output. The second part involves optimising the speed of the algorithm, by any means possible, while still returning an optimal solution.

Time: 4 days

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. This is a critical path as if we don't finish this in time testing will be delayed.

Dependencies: To be done after input data converted into a DAG (2.3) and output can be displayed (2.4). In parallel with output displayed.

2.6 Visualization implemented (Final Milestone)

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. 2 people should work on it on 2 different computers since no one has knowledge of this task, more research is required.

Dependencies: Need to implement algorithm (2.5). Done in parallel with algorithm parallelized.

2.7 Algorithm parallelized (Final Milestone)

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: 6 days

Resources: 2 people to 3 people since this is potentially a difficult task, pair programming will most likely be used. This task will be done in parallel with the visualisation part.

Dependencies: Need to implement algorithm (2.5). Done in parallel with visualization implementation.

2.8 Code documented (First Milestone and Final Milestone)

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. This is for general understandability.

Time: 1 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: Depends on live visualization tested (3.5), parallelization speed tested (3.4) and optimized and sequential speed tested and optimized (3.3).

3 Testing and Bug Fixes

3.1 Valid Schedule assessed (First milestone)

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: 4 days to write test cases alongside algorithm implementation

Resources: 1 person does this task. Everyone does one testing task in parallel.

Dependencies: Assessment will be done after the algorithm implementation (2.5). It will be done in parallel with the optimal schedule assessment (3.2).

3.2 Optimal schedule assessed (Final Milestone)

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: 3 days to write test cases and develop

Resources: 1 person does this task.

Dependencies: Assessment will be done after the algorithm implementation (2.5). It will be done in parallel with the valid schedule assessment (3.1).

3.3 Sequential speed tested and optimized (Final Milestone)

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: Sequential speed testing will be carried out after both schedule assessments (3.1 and 3.2).

3.4 Parallelization speed tested and optimized (Final Milestone)

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

Time: 2 day

Resources: 1 person to write test cases.

Dependencies: Parallelized speed testing will be carried out after the algorithm is parallelized (2.7).

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: 2 days.

Resources: 1 person.

Dependencies: This has to be done after implementing visualization (2.6).

4 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: 30 min

Resources: 1 person (The repository creator)

Dependencies: Start of project

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: 1 days

Resources: 1 person

Dependencies: Must be done after GitHub repository is created (4.1) and the project plan is created (1.3)

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: 4 days

Resources: Everyone will focus on the report

Dependencies: Can work on throughout project, but the final demo should be finished to complete the report. To be specific, this means after the third party software is packaged into the jar file (4.5).

4.4 Individual peer evaluation completed

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

Time: 1 hours

Resources: Everyone does it individually

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 the program is fully documented

4.6 Third party software documented

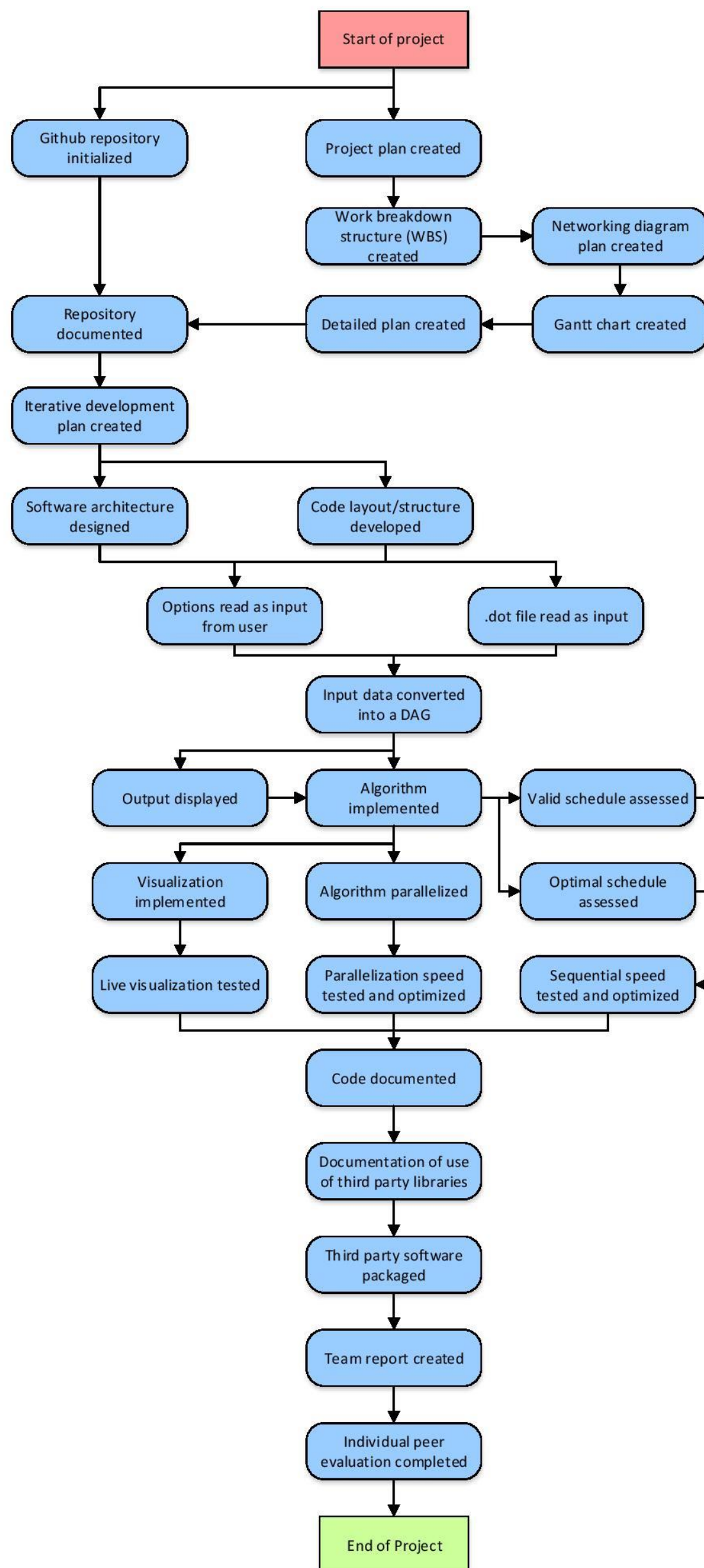
The process of ensuring that all third party libraries are documented and kept separate from code that we wrote. Where possible, this will be done through changes in the directory structure of the project.

Time: 1 day

Resources: At least one person, but ideally the whole team will review all code.

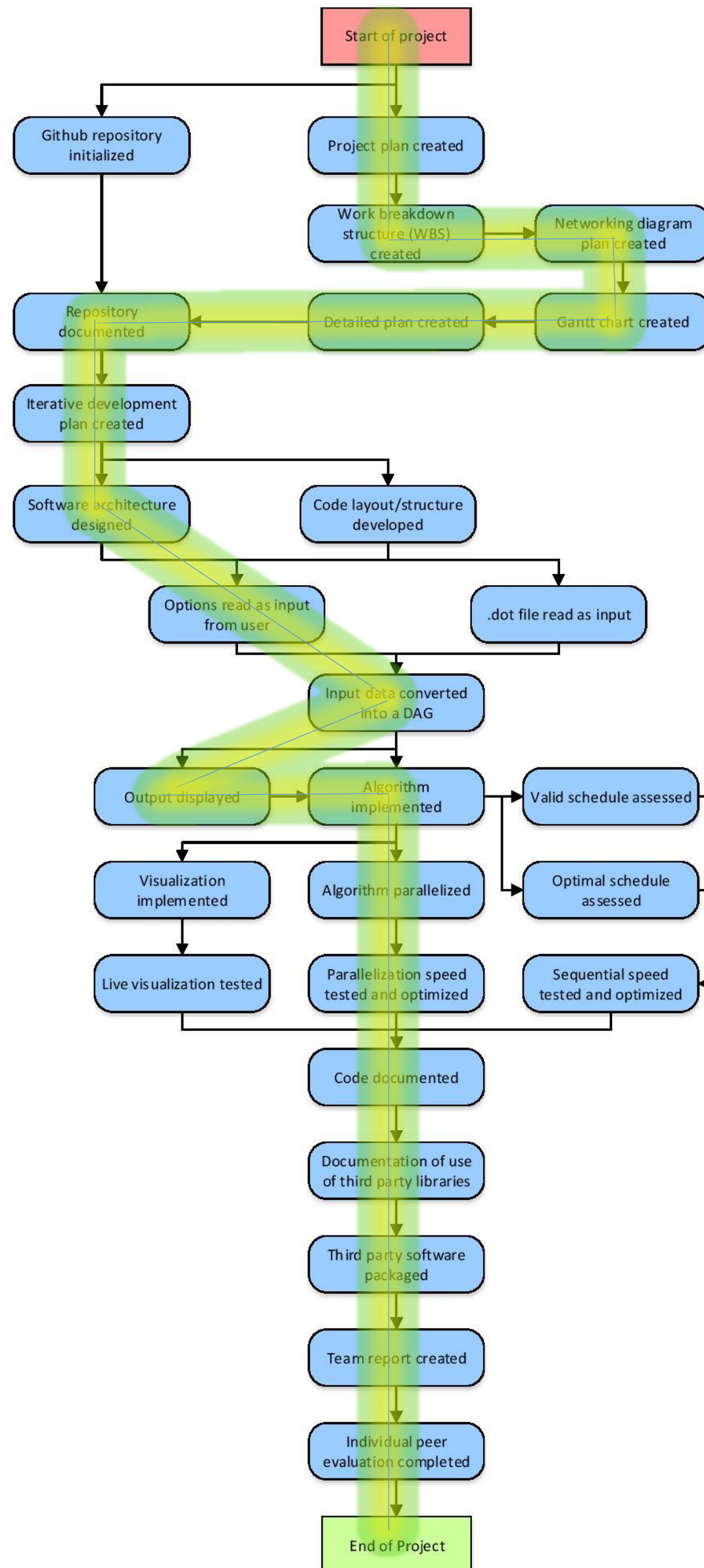
Dependencies: To be completed after code documentation (2.8).

Project Network



Critical path

Project Network critical path



Longest path assessment

As highlighted by the marked network diagram showing the critical path, the overall time required is ~33 days (32.6 exact) starting at 25th July to completion date on 26th August (33 days). This is done by allocating an upper bound value for the time required to do a particular task based on estimates. The path was calculated based on the time figures shown in the detailed explanation of each task and is complimented with the Gantt Chart.

Using those figures a critical path was calculated which is the path where if an issue occurs, it would have a large effect on the project. As such, detailed planning and care is required at those nodes (tasks) along the path. This is to ensure that the entire project can be completed before the deadline and in the best possible submission.

Even though the time of the critical path is approximately the same as the project duration, the times selected were to fit the entire project while giving the maximum estimated time to complete the relevant task.

Only the estimated time to complete a task was used to find the critical path as number of team members working (people) could fluctuate based on the workload or if tasks could be done in parallel.

Need times, resources and estimated time for each task can be found in the List of tasks explanation.

Working

[Project plan created, 10 days]
[Repository documented, 1 days]
[Iterative development plan created, 1 days]
[Software architecture designed, 2 days]
[Options read as input from user, 0.0833 days]
[Input data converted into DAG, 0.208 days]
[Output displayed, 0.0833 days]
[Algorithm implemented, 4 days]
[Algorithm parallelized, 6 days]
[Parallelization speed tested and optimised, 2 days]
[Code documented, 1 days]
[Documentation of use of third party libraries, 1 days]
[Third party software packaged, 0.167 days]
[Team report created, 4 days]
[Individual peer evaluation completed, 0.0417 days]
= 32.5833 days

Gantt Chart

