PROJECT 1
PLAN DOCUMENT

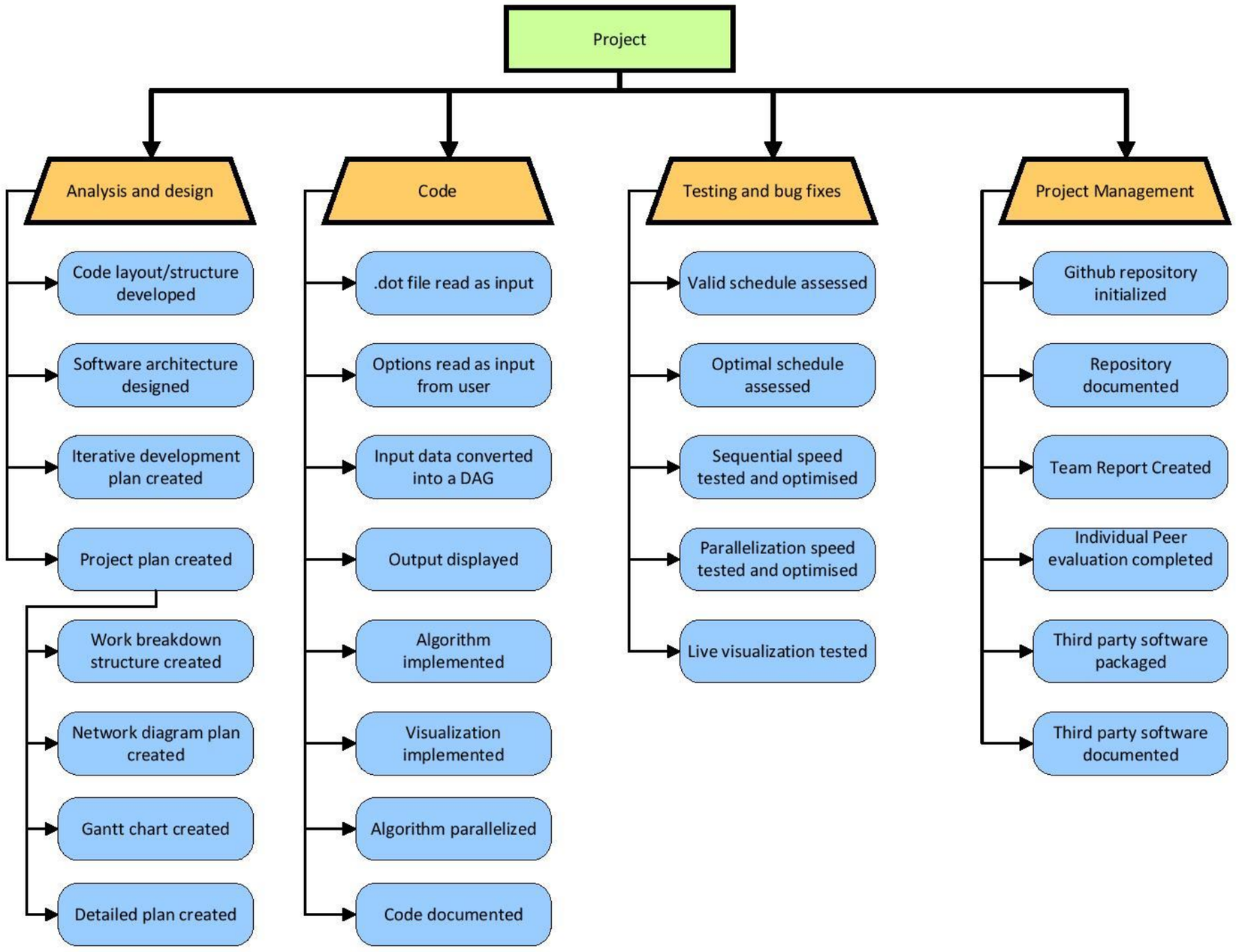
Group 6

Members

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

Contents

Work Breakdown Structure



List of Tasks

1 Analysis and Design

1.1 Code Layout/Structure Developed

Coding conventions such how to name variables and classes. This will allow the code to be understood when reviewing 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 day.

Resources: 2 people will 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 well-designed program architecture is essential to understanding the whole structure of the program. When we build, we will stick to this, so the code structure will remain clean This will be demonstrated using UML diagrams.

Time: 2 days.

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 days
- Gantt Chart created – 2 days
- Detailed plan created – 1 day

Planning the project helps reduce scope creep and guides us to finish the project in a timely manner. It allows us to organize time and resources in the most efficient way possible, so that no person is idle or overworked.

Time: 10 days.

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: None.

1.4 Iterative Development Plan Created

Ideally the foundations of the project will be complete within a week meaning that the software will output some valid schedule. However, it does not need to be optimal at this stage. We will make use of week-long iterations so that we can clean and improve existing code in small bursts to lead up to a finished program. An iteration involves fixing errors, certain classes, and methods to improve performance, as well as implementing new functionality. This step refers to the planning of such an iterative strategy.

Time: 1 day.

Resources: 2 people will be working on this in parallel with repository documentation. We will be using Trello as a means of organizing our tasks for agile development.

Dependencies: Done after Github Repository Documented (4.2).

2 Code

2.1 .dot File Read as Input

It is critical that this part is correct as any errors here will propagate throughout the entire program. 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 Software Architecture Designed (1.2) and Code Layout/Structure Developed (1.1), and in parallel with Options Read As Input From User (2.2).

2.2 Options Read As Input From User (Final Milestone)

Adjusting the main method to allow parameters to be passed in for things such as the visualization interface, or allowing the user to choose between parallel and serial operation. 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 (2.1).

2.3 Input Data Converted Into a DAG

When the input is read, the data needs to be transformed into a usable digraph data structure.
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 the same layout as described in the project brief. Creating a .dot file from the digraph data structure, as fast as possible, ideally O(n).
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 Converted Into a DAG (2.3). In parallel with Algorithm Implemented (2.5).

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, and deciding whether to use the Branch and Bound or A* algorithm, 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). In parallel with Output Displayed (2.4).

2.6 Visualization Implemented (Final Milestone)

Visualization of the code working to allow our client to see how the algorithm is running. We will use 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 too severely.

Time: 3 days.

Resources: 1 or 2 people. 2 people should work on it as no one has knowledge of this task, so more research will be required.

Dependencies: Need Algorithm Implemented (2.5). Done in parallel with Algorithm Parallelized (2.7).

2.7 Algorithm Parallelized (Final Milestone)

Initially the first completed part of the project would run sequentially, that is, on one processor. 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.

Dependencies: Need Algorithm Implemented (2.5). Done in parallel with Visualization Implemented (2.6).

2.8 Code Documented (First Milestone and Final Milestone)

Documenting the code of the whole software architecture. Ensuring that all team members followed the same conventions. Any complex sections should be commented, and larger, simpler sections should have comments illustrating what they do. All classes and methods need to have a formal Javadoc created. This is to aid general understanding.

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 and Optimized (3.4) 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.

Resources: 1 person writes the test cases. Testing may be done by the person implementing other code.

Dependencies: Assessment will be done after Algorithm Implemented (2.5). It will be done in parallel with Optimal Schedule Assessed (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 schedule created must be optimized by some algorithm, and will be tested using JUnit test cases.

Time: 3 days.

Resources: 1 person.

Dependencies: Assessment will be done after Algorithm Implemented (2.5). It will be done in parallel with Valid Schedule Assessed (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.
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 days.
Resources: 1 person.
Dependencies: Parallelized speed testing will be carried out after Algorithm Parallelised (2.7).

3.5 Live Visualization Tested

A visual test to see if the interface is showing the correct graph with the appropriate measurements of search progress and node structure. This is done in order to check if the visual function is working as intended and that the optimal solution is the one being displayed.
Time: 2 days.
Resources: 1 person.
Dependencies: This has to be done after Visualisation Implemented (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 minutes.
Resources: 1 person.
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: 1 day.
Resources: 1 person.
Dependencies: Must be done after GitHub Repository Initialised (4.1) and Project Plan Created (1.3).

4.3 Team Report Created

Overall assessment of group project, commenting on methods and tools 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.
Dependencies: Can work on it throughout project, but the final demo should be finished to complete the report. To be specific, this means after Third Party Software Packaged (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 hour.
Resources: Everyone.
Dependencies: After Third Party Software Packaged (4.5), the development will be complete, so the report can be written.

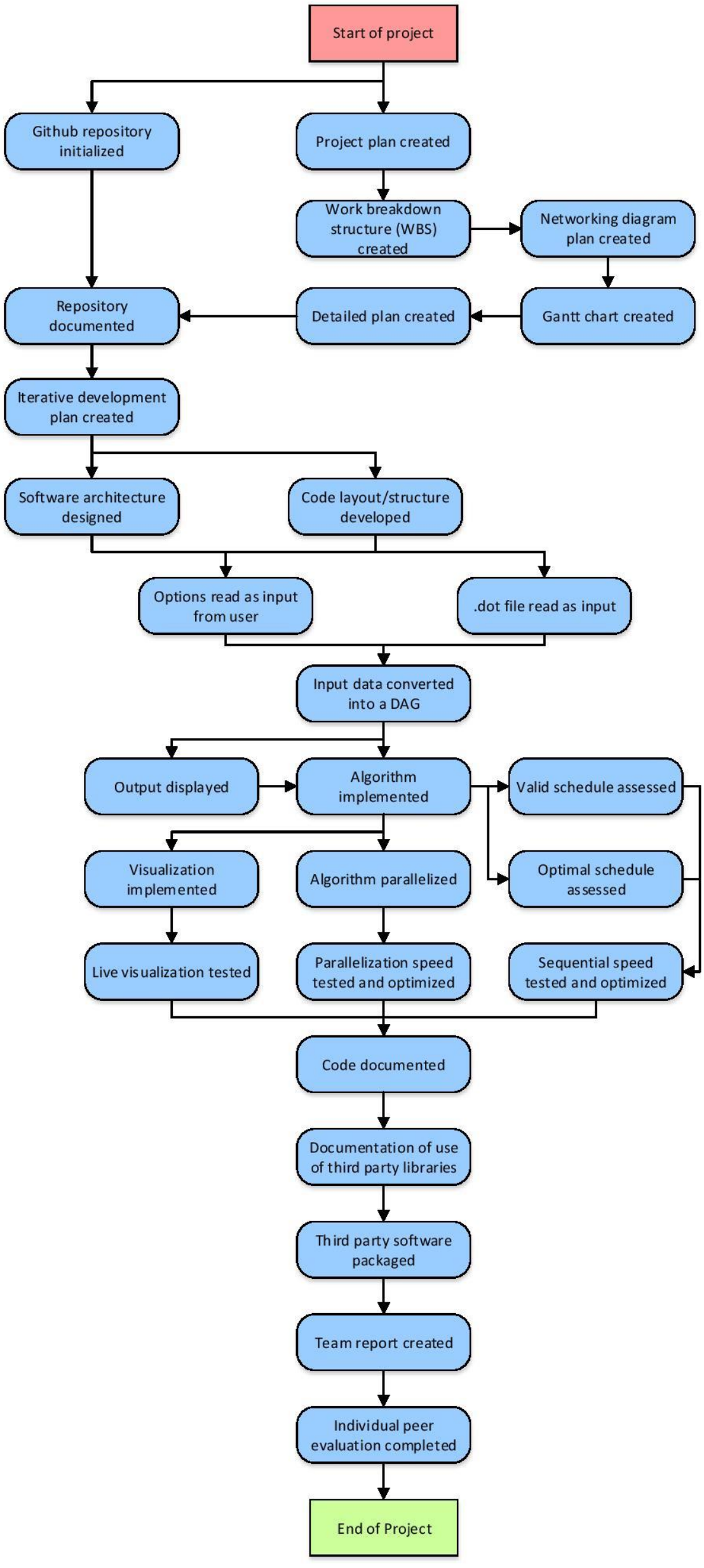
4.5 Third Party Software Packaged

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: After Third Party Software Documented (4.6).

4.6 Third Party Software Documented

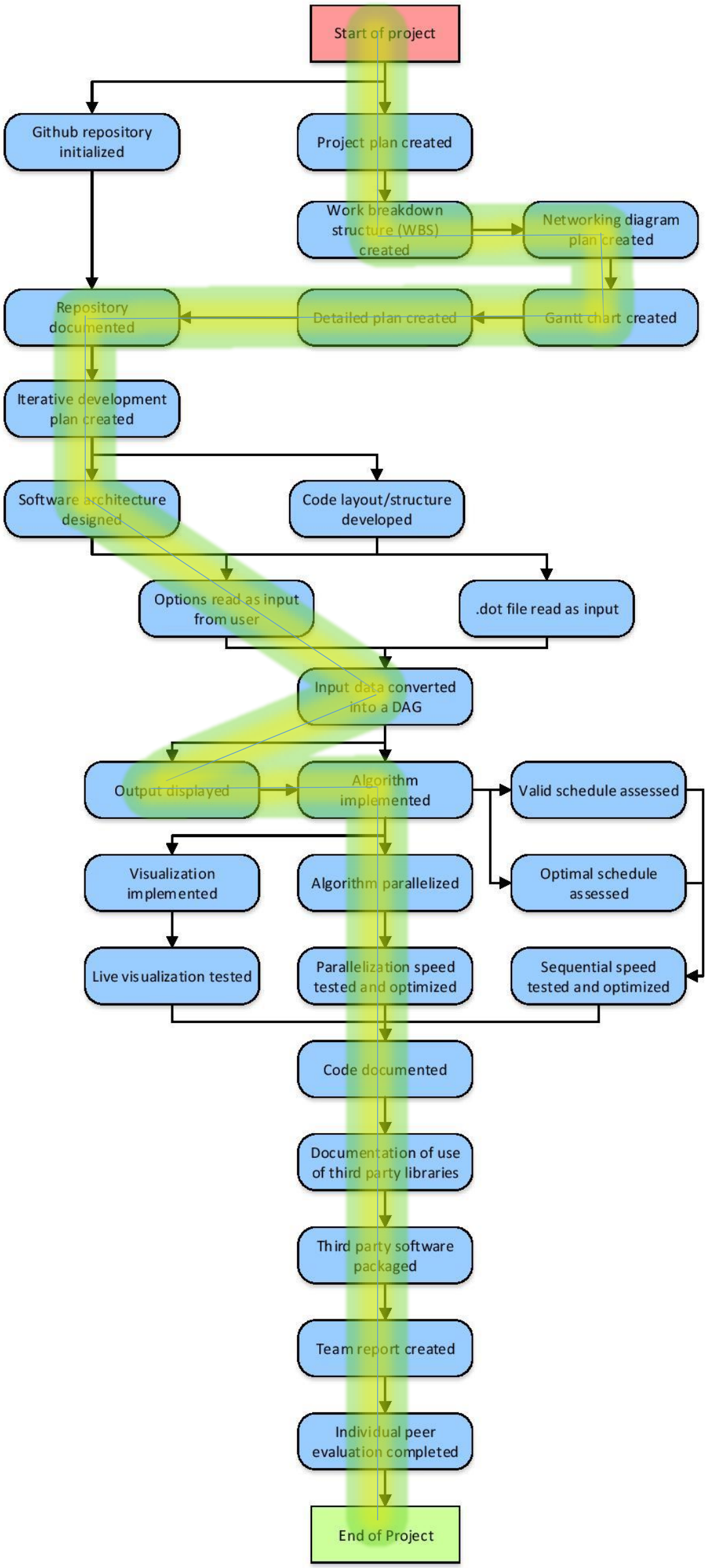
The process of ensuring that all third party libraries are documented and kept separate from code that we wrote. Where possible, this will be done through changes in the directory structure of the project.
Time: 1 day.
Resources: At least one person, but ideally the whole team will review all code.
Dependencies: To be completed after Code Documented (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 manner.

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 the 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

Task Name	Duration	Start	Finish
Project Planning	**10 days**	**Mon 25/07/16**	**Fri 5/08/16**
Github Repository created	30 mins	Mon 25/07/16	Mon 25/07/16
Work Breakdown Structure	1 day	Mon 25/07/16	Mon 25/07/16
Network Diagram Plan	2 days	Tue 26/07/16	Wed 27/07/16
Gantt Chart	2 days	Fri 29/07/16	Mon 1/08/16
Detailed plan created	1 day	Tue 2/08/16	Tue 2/08/16
Repository Documented	1 day	Wed 3/08/16	Wed 3/08/16
Iterative development plan created	1 day	Wed 3/08/16	Wed 3/08/16
Code layout/structure developed	1 day	Thu 4/08/16	Thu 4/08/16
Software Architecture Design	2 days	Thu 4/08/16	Fri 5/08/16
Coding First Milestone	**6 days**	**Fri 5/08/16**	**Fri 12/08/16**
Options read as input from user	2 hrs	Sat 6/08/16	Sat 6/08/16
.dot file read as input	4 hrs	Sat 6/08/16	Sat 6/08/16
Convert input data into DAG	5 hrs	Sat 6/08/16	Sat 6/08/16
Display output	2 hrs	Sat 6/08/16	Sat 6/08/16
Branch and Bound Algorithm Implemented (Sequential)	4 days	Sun 7/08/16	Wed 10/08/16
Testing Valid Schedule	4 days	Tue 9/08/16	Fri 12/08/16
Optimizing Final Milestone	**11 days**	**Fri 12/08/16**	**Fri 26/08/16**
Optimal Schedule Assessed	3 days	Fri 12/08/16	Tue 16/08/16
Algorithm Parallelized	6 days	Fri 12/08/16	Fri 19/08/16
Visualisation Implemented	3 days	Fri 12/08/16	Tue 16/08/16
Live Visualisation Tested	2 days	Tue 16/08/16	Wed 17/08/16
Parallelization Speed tested and optimized	2 days	Sat 20/08/16	Sun 21/08/16
Code Documented	1 day	Mon 22/08/16	Mon 22/08/16
Documentation of use of 3rd party libraries	1 day	Mon 22/08/16	Mon 22/08/16
Third party software packaged	4 hrs	Mon 22/08/16	Mon 22/08/16
Team report	4 days	Tue 23/08/16	Fri 26/08/16
Individual Peer Evaluation	1 hr	Fri 26/08/16	Fri 26/08/16