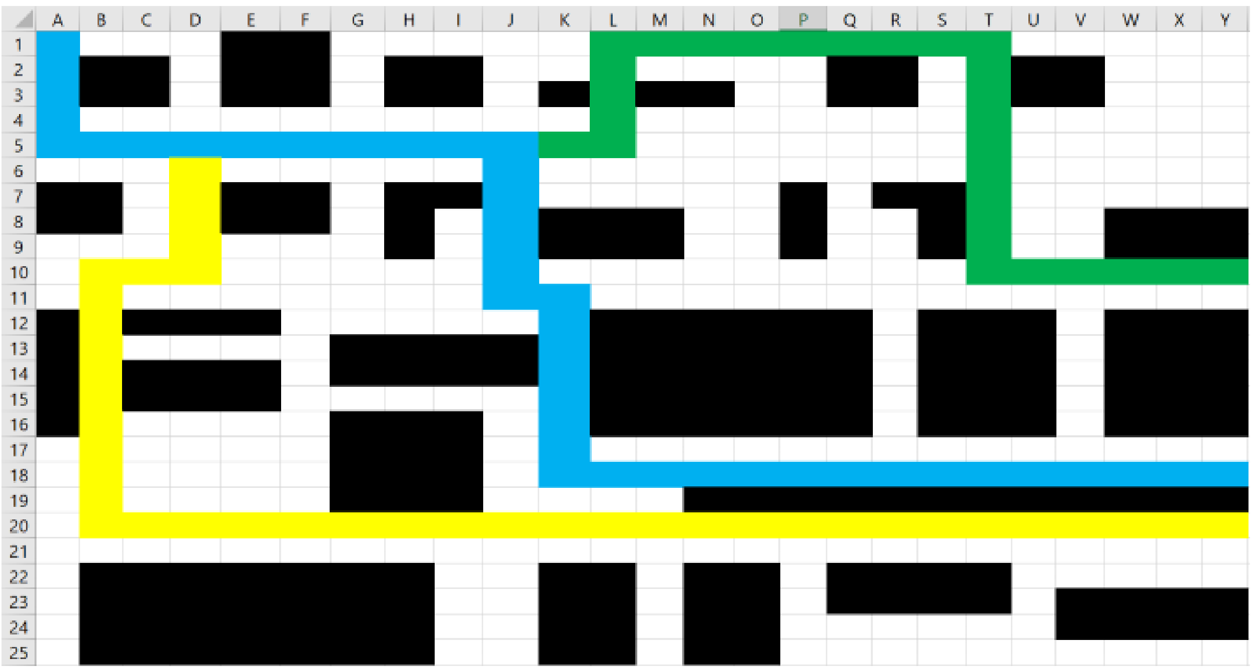
SFT221 – Project

Learning Outcomes

- Design and build the solution to a small problem,
- Design the testing for the problem.

Project Description

You have been hired by a local delivery company that has three different trucks that deliver on three different routes in your part of the city. The map of your city is represented as a 25 by 25 square grid as shown in the diagram below.



Our offices are located at the square 1A. From that location our three different delivery vans start out initially on a common path and then branch out to cover different parts of the neighborhood. The paths taken by our delivery trucks are shown in blue, yellow, and green. Initially, all trucks start out and follow the blue path and then the green and yellow trucks branch off from the path taken by the blue trucks. The black rectangles represent buildings to which packages can be delivered. The white areas are empty space and the blue, yellow, and green areas represent the paths taken by our three delivery trucks. All addresses are specified in terms of the row number and column letter to which a package should be delivered.

All our trucks are the same size and can hold up to 1000 kilograms of cargo. The trucks are also capable of carrying 36 cubic meters of boxes. The boxes that we use for shipping come in 1/4 cubic meter, 1/2 cubic meter, and 1 cubic meter sizes. All our boxes are square, meaning that they have the same dimensions on all sides.

When a customer comes in with a shipment, they specify:

- The weight of the shipment in kilograms,
- The size of the box required in cubic meters,
- The destination of the box in terms of a building specified by a row number and column letter that is within the black rectangle represented by the building.

When a shipment comes in, your job is to find a truck which is big enough to hold the shipment as well as finding a truck which is going to go as close as possible to the destination of the package. The trucks can divert slightly from their assigned routes and pass through any of the white parts of the grid to deliver the package. You should always put the package in the truck that we'll have to divert the least distance to deliver the package. When considering the distance a truck must divert from its route to get to the destination, you must remember the trucks cannot drive through any of the black buildings on the map. If two trucks are the same distance away from the destination, you should put the package in the truck which is less full.

A truck is full when it hits either its maximum weight or maximum volume – whichever is reached first. For example, if a truck already has 900 kilograms in it but only has 10 cubic meters of boxes then the limiting factor must be taken as the weight. If another truck has 30 cubic meters of boxes but only 200 kilograms of cargo, then you must assume that it is limited on space rather than weight. When you compare two trucks to see which one has the most space remaining, you should look at the limiting factor for each truck as a percentage and compare the percentages.

Measuring Distance

While it is relatively easy to determine the truck that has the most room or weight available for a package, it is less obvious how to determine which truck comes closest to the destination for a particular package. In some cases, it doesn't really matter since they will be equidistant.

Consider the case where a package needs to be delivered to 7F. This location is actually on the route of two of our trucks. Since the green and blue routes overlap at this point it does not really matter whether we assign it to a blue or a green truck and that must be determined by the truck which has the most available space. On the other hand, it is very close to the route of the yellow truck. To determine whether we should put it in a yellow truck or one of the blue or green trucks we must measure the distance from the nearest point on the yellow trucks route to destination and the distance from the blue and green trucks to the destination. Looking at the problem visually, we can see that the blue and green trucks come within two squares of the destination but the yellow truck comes within three squares of the destination. Therefore, since the blue and green trucks come equally close to the destination, the truck in which the object will be placed is based upon available weight and volume.

Now, let's look at a more difficult problem. Consider the case where we want to make a delivery to 8P. This is near to both the blue line as well as the green line, but it is not obvious which one is the closest in order to find which was the closest we have to do a little bit of Euclidean geometry. As you might remember from school, the distance between any two points is the square root of the sum of the two sides of the triangle. The nearest point on the blue route is at 11K and the distance to 8P is $\sqrt{3^2 + 5^2}$ which equals 5.83. The green truck might be closest at 6 T or it might be closest at 10T. To figure out which is closest, we use the technique we just used above to calculate the distance for each. They distance from 6T 8P is 4.47 and the distance from 10T to 8P is exactly the same. Of course, 8T is only 4 spaces away from 8P, making it the closest in terms of Euclidian distance. A human can see this but the human also sees that there is a building in the way and the truck needs to go around it, making that route even longer.

The Shortest Path Algorithm

One of the problems we need to solve is how to find the shortest path between two points given that you cannot go through a building. One solution to this problem is what is called the A* algorithm. The A* algorithm users the Euclidean distance from where you are on the path to the destination as a heuristic as to which way to go at any point.

Consider being at 8T and wanting to go to 8P. This is the shortest Euclidian distance to the destination, but might not be the shortest when you consider not being able to go through buildings. When you are at 8T, you can go to 7T, 7U, 8U, 9T or 9U since these all touch on 8T and are not part of a building. What we do next is to measure the Euclidian destination from each of these squares to the destination, as shown in the table below.

Start Square	Distance to 8P
7T	4.12
7U	5.1
8U	5
9T	4.12
9U	5.1

From this table you can see that both 7T and 9T are equidistant to destination and our shortest distances. The algorithm says you always pick the shortest distance and go to that square. Since the two squares are the same distance, it does not matter which one we pick. Therefore, we will randomly select 7T as our destination.

Once we get to 7T, we need to go through the same process again. We select all the squares it is possible to move to and then calculate the distance to the destination square and move to the square which is closest to the destination. We repeat this process until we reach the destination. If we count the number of squares we move along the way, we will find the total distance that it takes to get to the destination.

One of the big problems with the shortest path algorithm is that it can get stuck. It can get stuck either at the edge of the map or in a corner of a building. You know you are stuck when you have tried every direction in which you can move forward, and you cannot reach the destination. Remember, you cannot go back along the path you came, as this can result in going in circles. At this point, you will simply say that the destination cannot be reached and mark that as not an eligible point to divert from the route to make a delivery.

The Overall Algorithm

The overall algorithm will

- follow the route for each of the trucks.
- At every square on the route for each truck it will calculate the Euclidean distance to the destination.

- It will select the minimum Euclidean distance for each of the trucks and then calculate the shortest path from each of those positions to the destination.
- In the event one of the trucks cannot find a path to the destination, that truck will not be used for the delivery and one of the other trucks will be used.
- Finally, it will select the truck which has the shortest path from the nearest point on its route to the destination and attempt to add the package to that truck.
- If that truck cannot hold the package, it will try to put it in the truck that is next closest to the destination.
- If no truck can take the package, it will be placed in storage at the depot until the trucks return empty and it will be shipped out the next day. It will print the message “Ships tomorrow”.

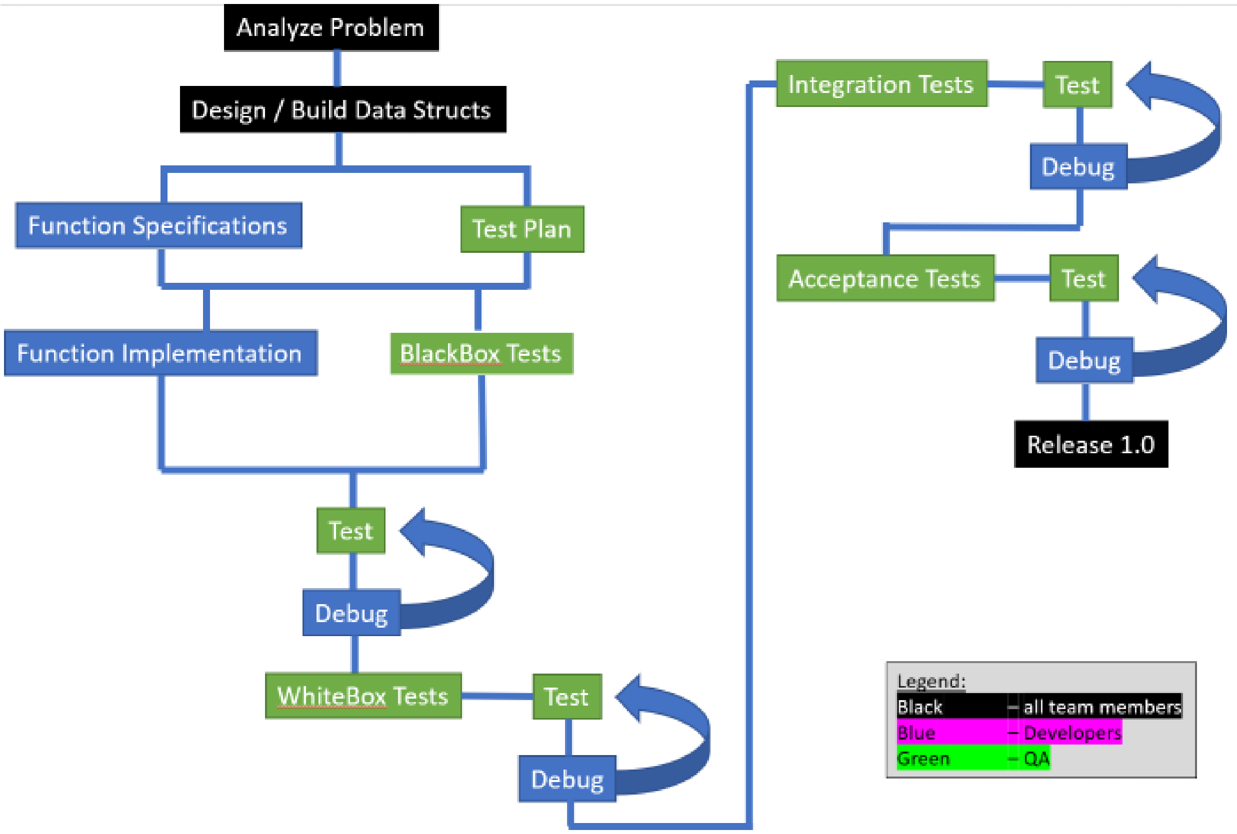
Our algorithm will run for one day. This means we accept items until all trucks are full or items stop arriving. Once this happens, the trucks will be dispatched but that is not the responsibility of the program. All the program needs to do is:

- Decide which truck the package will be placed in,
- Print out where the truck will deliver the package,
- If the truck needs to divert to deliver the package, you will print the path to divert.

Sample Output

```
=================
Seneca Deliveries
=================
Enter shipment weight, box size and destination (0 0 x to stop): 20 .5 28x
Invalid destination
Enter shipment weight, box size and destination (0 0 x to stop): 20 2 12L
Invalid size
Enter shipment weight, box size and destination (0 0 x to stop): 1005 .5 12L
Invalid weight (must be 1-1000 Kg.)
Enter shipment weight, box size and destination (0 0 x to stop): 20 .5 12L
Ship on BLUE LINE, no diversion
Enter shipment weight, box size and destination (0 0 x to stop): 200 1.0 8Y
Ship on GREEN LINE, divert: 7T, 7U, 7V, 7W, 7X, 7Y, 8Y
Enter shipment weight, box size and destination (0 0 x to stop): 500 1.0 8Y
Ship on GREEN LINE, divert: 7T, 7U, 7V, 7W, 7X, 7Y, 8Y
Enter shipment weight, box size and destination (0 0 x to stop): 500 1.0 8Y
Ship on BLUE LINE, divert 18V, 17V, 16V, 15V, 14V, 13V, 12V, 11V, 10V, 9V,
8V, 7V, 7W, 7X, 7Y, 8Y
Enter shipment weight, box size and destination (0 0 x to stop): 0 0 x
Thanks for shipping with Seneca!
```

Deliverables



This project has multiple milestones and will follow a development process approximately the same as indicated above. Since the process is agile, it will deviate from the above process whenever it makes sense to do so.

During the project, teams are expected to

- use the Git repository to store all work produced by the team.
- Document code with a comment at the top of each data structure and function which not only states the purpose of the data structure or function but also the author(s). Changes to code should include a comment saying who made the change and why.
- Add comments when items are committed to Git indicating what is being added or changed and why.
- Create an issue in Jira every time new work needs to be done,
    - Move the issue across the Kanban board as its status changes
- Add comments to the issue as work progresses,
- Meet one or more times per week to discuss the project,
- Team members should select work items they want from the Kanban board and assign it to themselves. It is the responsibility of each team member to ensure that they do their fair share of the work.

Marking

Marking is split into individual and group sections. The individual marks are for your contribution to the project. You could lose marks for not having you name on a fair share of the work, missing team meetings, not having your material ready on time etc. You are also marked on your team work, which is another measure of timeliness and collaboration with the group. You could lose marks for not selecting work to do, not being on time with submissions, not helping with administrative duties like updating Kanban or updating Git.

Group marks are based on the overall results of the group. The "Meets Deadlines" part is how well the group overall meets its deadlines. This involves individuals meeting deadlines and the group having everything complete and well done by the deadline. You could lose marks for having all the work done just before the deadline, having part of the work missing by the deadline or poor-quality work on the deadline.

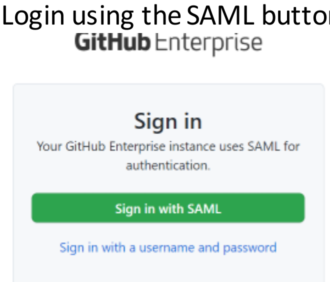
Milestone 1

In this phase of the project you will:

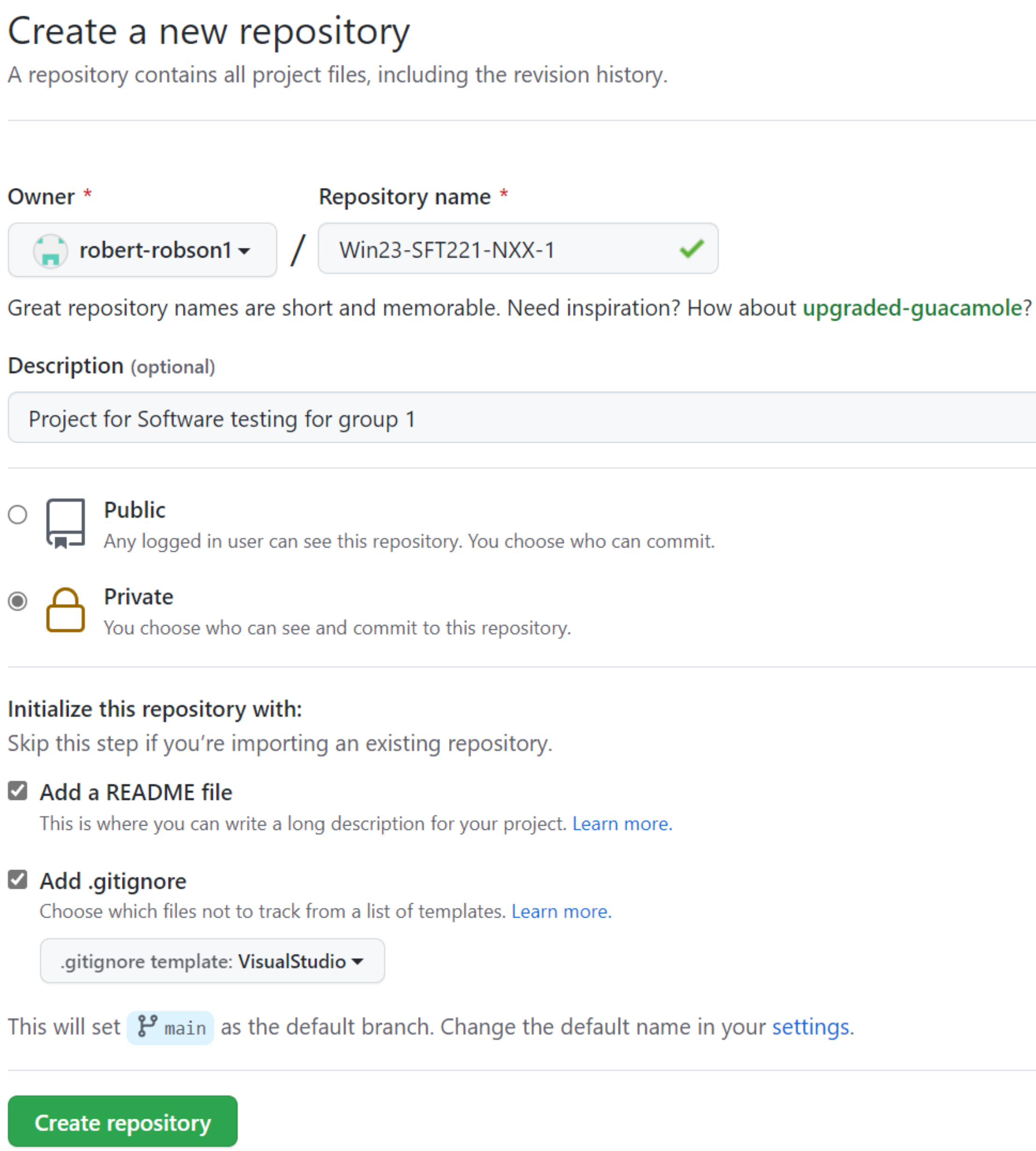
- Setup teams of about 5-6 developers (7 is too large)
- Write and sign a team contract
- Create a GIT account
- Create a Jira account

Create GitHub Account

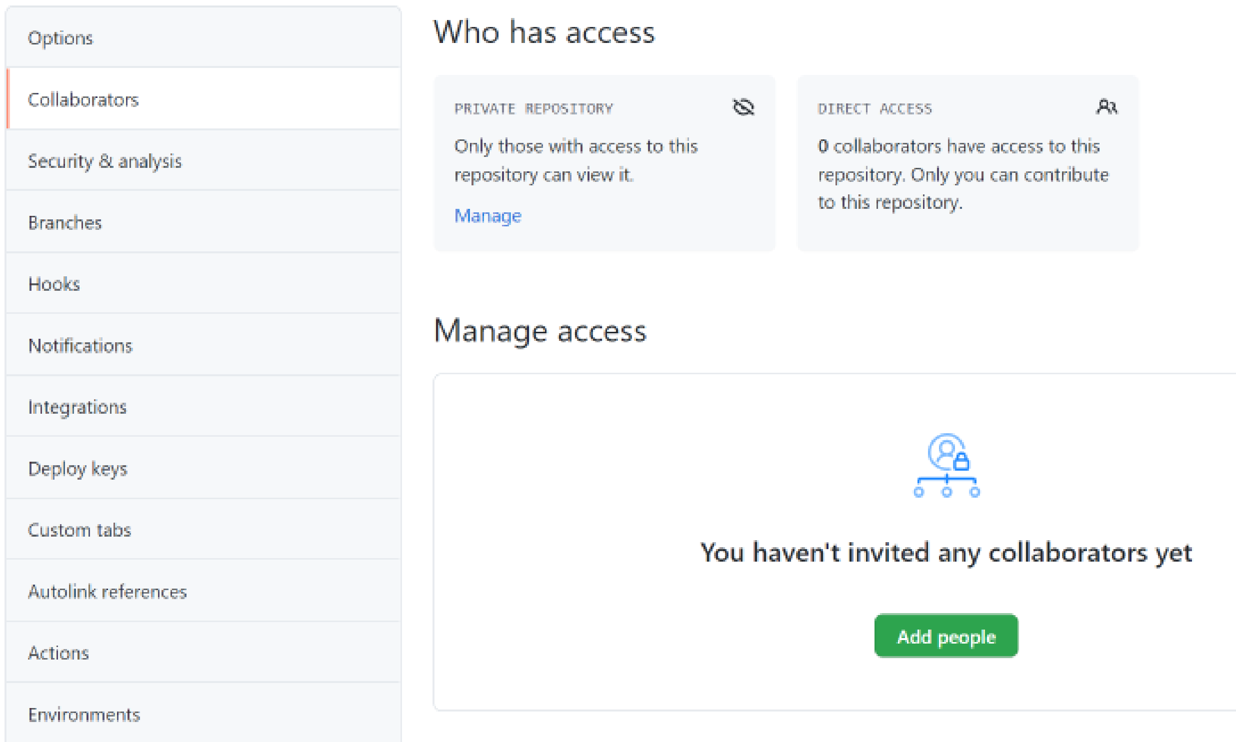
1. Sign into the Seneca VPN
2. Go to https://github.senecacollege.ca
3. Login using the SAML button



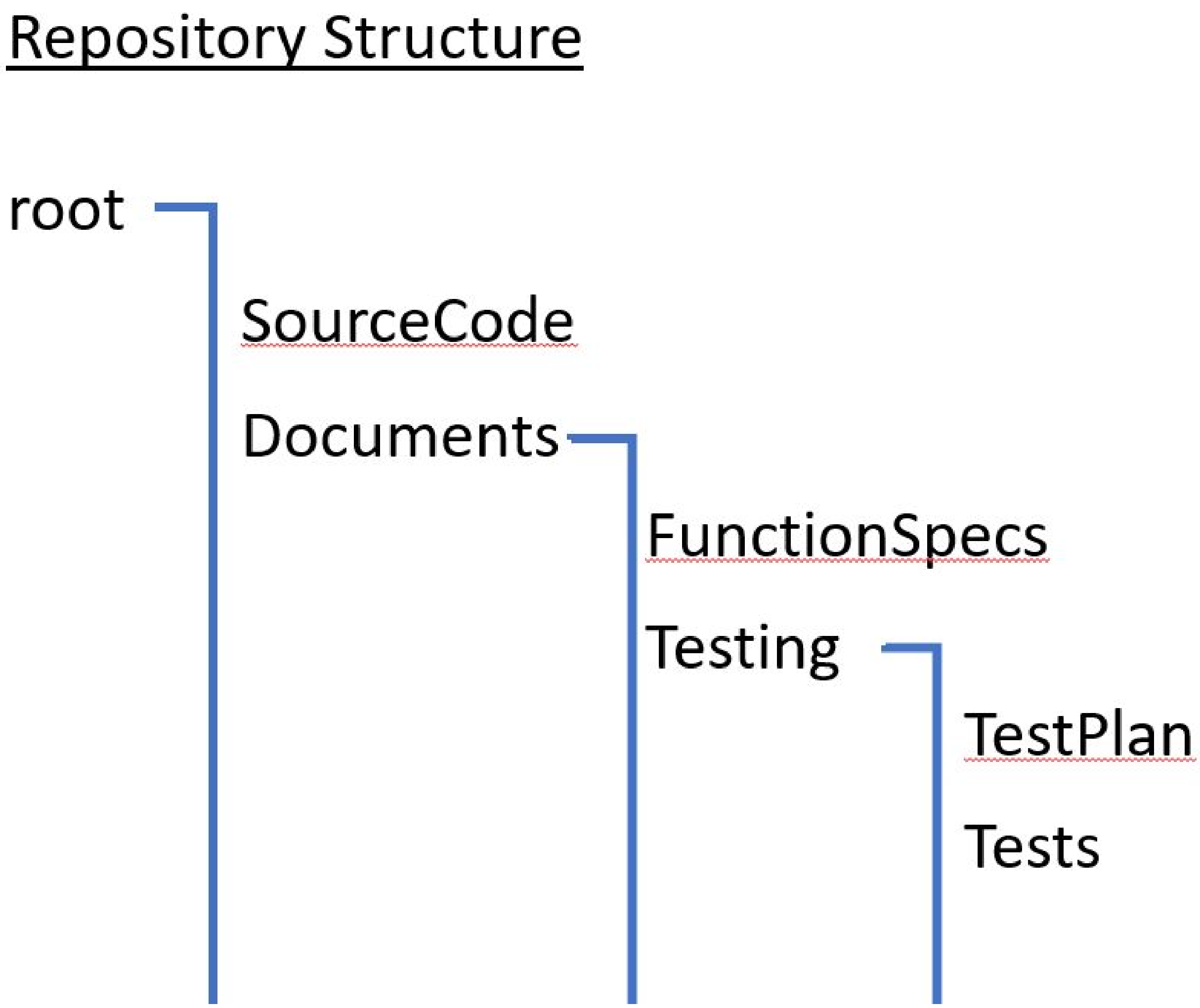
4. This will set up GitHub for each student. One (and only one) member of the group will be selected as the group leader for GitHub. Your professor will create an empty repository following the naming pattern: Term-CourseID-SectionID-Group. For example, I used Win23-SFT221-NXX-1 which means it is for the Winter 2023 term for the course SFT221 for section NXX and this is group 1. The repository must be private so only your group members can see it. Check to add a readme file and a .gitignore file for VisualStudio. Your professor will add the selected group member as the admin for the repository. Once set up by the professor, the new admin can invite the rest of the group members.



5. Add your fellow group members and your professor to the project by going to the settings and selecting **Add People**

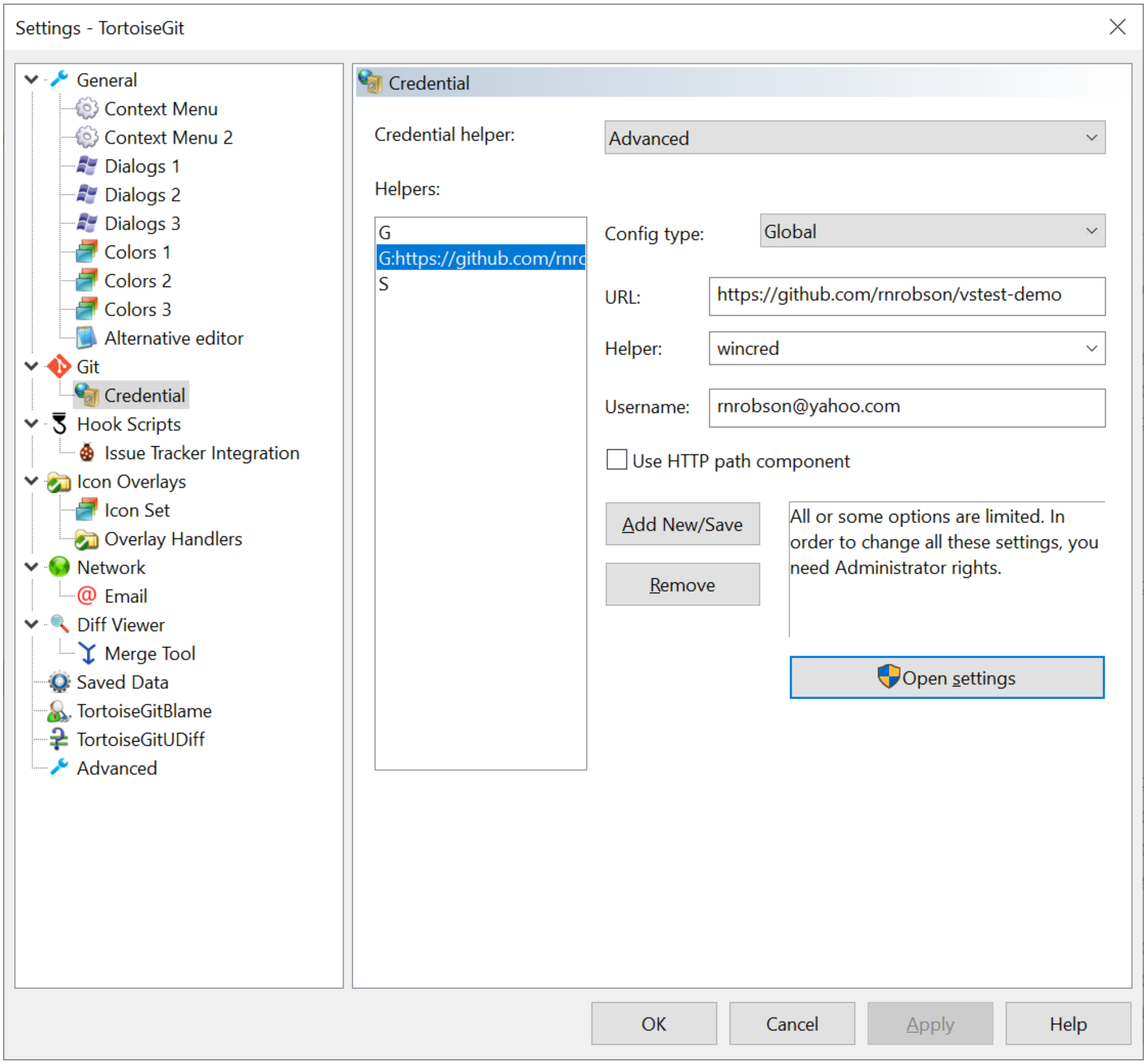


6. Install tortoiseGit on your windows computer which can be downloaded from https://tortoisegit.org.
7. The group leader should clone the group repository onto a local computer and set up the following directory structure with one readme.md file in each directory.



The group leader should then commit and push the changes to the remote repository.

8. Finally, GitHub is moving away from user names and passwords and requiring the use of access tokens. These are strings you can use in place of a password. There are instructions on how to create them here:
   [Tokens and Keys (sharepoint.com)]()
9. Then, you need to set up TortoiseGit to access your account. Right click on a file browser to show the TortoiseGit menu and select settings.

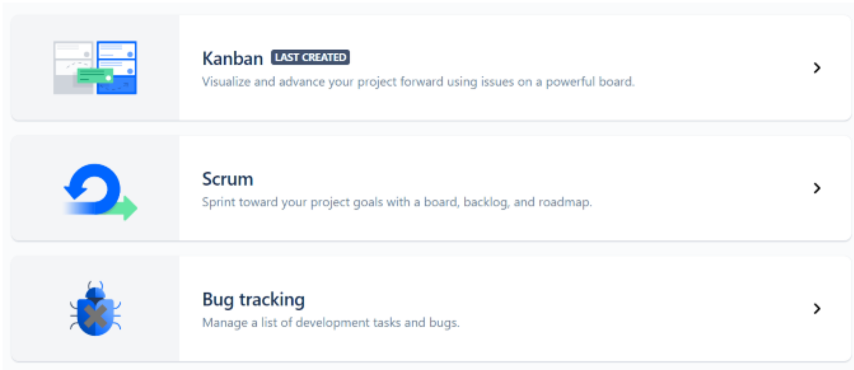


This is my example for GitHub.com. You will build a new Global credential for github.senecacollege.ca and fill in your user name. Use the **Add /New/Save** button to create the new credential and click **OK** at the bottom. Once you try to access the remote repository, it will prompt you for the token you were given, which you will use in lieu of a password. It will store this so that you only need to provide it again if it expires.

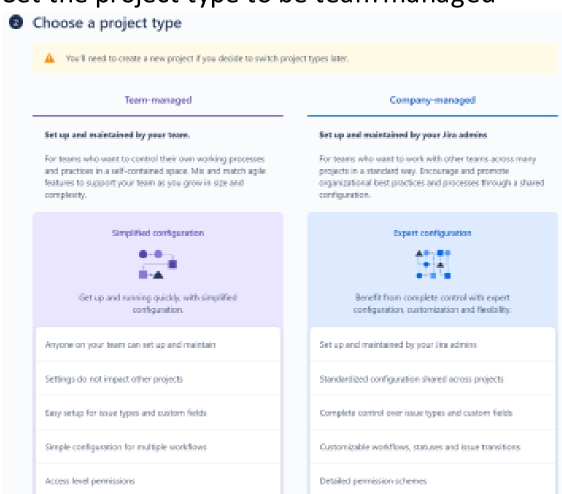
Set Up a Jira Account

1. All group members should go to https://www.atlassian.com
2. Select Sign In and then Sign Up for an Account
3. Go to Products | Jira Software and get a free account
4. The group leader should set up a new project

a. Create a new project and select Kanban



b. Set the project type to be team managed



c. Finally, set the project name to be the same as the Git repository you created.

d. Go to the project dashboard and click Add People icon to right of search box to add your teammates and professor to the project.

Deliverables Due at End of Lab

- Completed team contract
- Fully initialized Git repository
- Fully setup Jira project

Rubric

Individual	Work on contract, repository, or Jira project	10%
	Teamwork	10%
Group	Contract	15%
	Git Repository	35%
	Jira Project	30%

Milestone 2

Some of the software for the project has already been written for you and is available on Blackboard. You must use this in your project and every team should add it to the source code for their repository. Anything in the main is simply for demonstration purposes and can be replaced. The software you are being given has not been tested and you will need to test it.

You need to study the problem and the code provided for you and then:

- Add any new data structures you will require (this will require a thorough analysis of the problem and the existing software. You do not want to go back and modify it later if you can avoid it as it will slow the project.)
- Create a test plan for the project.

Deliverables Due at End of Lab

- An analysis of the problem (no written artifacts produced),
- A series of data structures created as header files and stored in the repository,
- A test plan stored in the repository.

Rubric

Individual	Data structures or work on Test Plan	15%
	Teamwork	10%
Group	Data structures (complete, correct and well-designed)	20%
	Test Plan (complete, well-written)	20%
	Git Usage (used properly with good structure)	10%
	Jira Usage (creates issues, tracks progress)	15%
	Meets Deadlines	10%

Milestone 3

In this milestone you will create issues to design the functions, design all of the functions you need to complete the project and store the specifications in the repository. As soon as the specifications start to be produced, you can start to design the blackbox tests (what they test, how to perform them and test data). Once tests are written, they can be implemented and added to the repository and any team members not otherwise busy can start to implement the functions. You will also build a function-test matrix that shows the blackbox tests for each function. This will be maintained through the testing cycle as new tests are added.

Deliverables Due at 23:59 4 Days after Lab:

- A set of function specifications stored in the repository,
- A set of blackbox tests as test documents with test data for the functions.
- Start writing blackbox test code and store in repository.
- Start implementing functions and store in repository.
- A function-test matrix added to the repository.
- Updated Jira project to show activities and progress.

Rubric

Individual	Work on functions or tests	15%
	Teamwork	5%
Group	Function Specs (documented, correct, complete, well-written)	20%
	Test documents (well-written, complete, good test data)	20%
	Test Code (well-designed, written and documented)	5%
	Git Usage (used properly with good structure)	10%
	Jira Usage (creates issues, tracks progress)	15%
	Meets Deadlines	10%

Milestone 4

This milestone will complete the implementation of the functions, the blackbox tests, execute the blackbox tests, and create new issues for any failed tests. Tests which passed can have comments added to the issue and it can be moved ahead on the Kanban board. Whitebox tests will be written and stored in the repository and then implemented and executed. The goal is by the end of the milestone all unit tests will have been run and the code debugged.

Test Automation

Now that we have tests to run, we need to automate the testing process. The goal of this automation is to ensure the code has passed the suite of tests before pushing the code to the repository. This will keep bugs out of the repository and eliminate the headaches which result from having a bug introduced into the repository.

To automate the testing, we need to install a script into the directory **.git/hooks** in your repository. Directories under .git are not checked into the remote repository so every team member on the project has to install his own hook. The hook must be stored in a file called **pre-push** in the **hooks** directory. The contents of the file will be like this:

```
#!/bin/sh
echo "Starting test runner..."

test_runner="C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\IDE\CommonExtensions\Microsoft\TestWindow\vstest.console.exe"
tests_dll="SimpleVSUnitTest\Debug\UnitTest1.dll"
failure_string="Failed:"

testout=`"$test_runner" "$tests_dll"`

if [[ $testout == *"$failure_string"* ]]; then
  echo >&2 "Unit test are not passing! Push aborted!"
            exit 1
fi
echo >&2 "All tests run successfully."
exit 0
```

This is the script set up for my machine and every team member will need to edit it to work on their own computer. You need to edit:

- **test_runner** – this is the location of the vstest.console.exe program installed by Visual Studio. Check the location and modify it if it is different on your computer.
- **tests_dll** – this is the location of the dll built for your test project. It is relative to the root of the project. Modify this to refer to the dll you want to test.

Once this is installed, every time you try to push to the remote repository, it will run the tests. If the tests fail, the push will be aborted. You will need to fix the code and them commit the changes and push again until the tests pass.

Deliverables Due at 23:59 4 Days after Lab:

- Implemented Functions
- Implemented blackbox tests (store in repo), executed (results in Jira) and debugged,
- whitebox tests written and stored in repository.
- whitebox tests implemented (store in repo), executed (results in Jira) and debugged.
- Updated function-test matrix stored to the repository.

Rubric

Individual	Work on tests, test execution or debugging.	15%
	Teamwork	5%
Group	Implemented Functions (well-designed, written and documented)	10%
	Whitebox tests (well-designed, written and documented)	20%
	Test Execution (performed, results recorded, issues created)	10%
	Debugging (Bugs fixed, documented, Jira updated)	10%
	Git Usage (used properly with good structure)	5%
	Jira Usage (creates issues, tracks progress)	15%
	Meets Deadlines	10%

Milestone 5

In this milestone, you should write, implement, and execute integration tests. Integration tests test how multiple functions work together to complete a task. Depending on what is being tested, you might be able to write unit tests to do the testing and automatically compare the results. In other cases, you might need to manually check the output to check it. This will all be stated in the tests where it discusses how they should be run.

As you update the function-test matrix, you will need to add a very brief description for each integration test so the matrix will clearly show what the tests are testing. Acceptance tests will be tested against actual user requirements and will list all the tests for each requirement.

Acceptance tests are the final tests and are largely aimed at showing the customer that the correct output is produced for different inputs. This will largely require manual testing.

Deliverables Due at 23:59 4 Days after Lab:

- integration tests written and stored in repository,

- integration tests written (store in repo), executed (results in Jira) and debugged.
- acceptance tests written and stored in repository.
- Updated function-integration-requirements-test matrix stored to the repository.

Rubric

Individual	Work on tests and debugging	15%
	Teamwork	5%
Group	integration tests (well-designed, written and documented)	15%
	acceptance tests (well-designed, written and documented)	15%
	Test Execution (performed, results recorded, issues created)	10%
	Debugging (Bugs fixed, documented, Jira updated)	10%
	Git Usage (used properly with good structure)	5%
	Jira Usage (creates issues, tracks progress)	15%
	Meets Deadlines	10%

Milestone 6

This is the final milestone where you will run the acceptance tests and fix any remaining bugs found. In addition, you will produce a testing report which lists all the tests conducted, the results and whether the bugs were fixed, and the final test passed. You will also review the test matrix to ensure every test has been performed and passed. You can change the colour of the test in the matrix to show it was run and passed. At the end, all tests in the matrix should have been passed.

The final test report can be tabular like this:

Function/acceptance/requirement	Test Run	Bugs Fixed	Passed
Distance	TF001	Did not handle negative coordinates	☑

Deliverables Due at 23:59 4 Days after Lab:

- Execute acceptance tests(results in Jira), and debug.
- Updated function-test matrix stored to the repository.
- Final Testing report listing tests conducted, bugs fixed and the final test passed.

Rubric

Individual	Work on tests, test matrix and final report.	15%
	Teamwork	5%
Group	Updated test matrix	15%
	Final test report	20%
	Test Execution (performed, results recorded, issues created)	10%
	Debugging (Bugs fixed, documented, Jira updated)	5%
	Git Usage (used properly with good structure)	5%

	Jira Usage (creates issues, tracks progress)	15%
	Meets Deadlines	10%