Interim Report

Software Engineering Project 3

The Shady Bunch

Caroline Hyland - 13409662
Karl O'Brien - 13529873
Mark McKenna - 13504023
Niamh Mc Eniff - 13343846

Project Specification

The overall goal of this assignment is to design and implement Java classes to represent a table that maps people to preferences and that supports a search for the best mapping from one to the other.

Our first step in completing this assignment was to create a *PreferenceTable* class and have our program take in the data from a tab separated file and store it in memory. Each line represented a student and their list of preferences so we decided storing each line in vectors would be the most efficient course of action.

Once this information was stored in memory we created another class that would take a vector and create an object *StudentEntry*. This object consisted of the student's name, list of preferences, an integer of the number of preferences and a boolean variable stating whether or not a student's preference was preassigned or not.

The *StudentEntry* class also included some other useful functions for later on in the project such as getting a random preference as well as adding a preference to the student's list if they hadn't filled out enough slots. After this we were able to add other useful functions into our *PreferenceTable* class, such as a method to return all of the *StudentEntry* objects.

As the overall aim of this project was to create a mapping of students to preferences our next job was to create the *CandidateAssignment* class that would link a S*tudentEntry* object to a particular preference. This object also included a variable which contained the previously assigned preference as to conveniently undo a mapping which lowered the total satisfaction of a solution.

An assignment's energy was calculated as the square of the assigned preference's position in its array plus one, so if the preference is located in position 2 the energy of that assignment would equal (2+1)^2

Lastly, our program needed a way to handle all the students' assignments together, and compute the energy of a complete mapping. This was done in our final class *CandidateSolution*. In this class, we created a hashtable to hold each of the *CandidateAssignment* objects. We could then iterate through this hashtable and calculate the total energy of the 52 assignments. We also had to include a penalty system in the event of two candidates being assigned the same project. This penalty added 1000 to the total energy. The program also allowed the user to see the fitness of a solution, the fitness being the energy multiplied by -1.

That marked the end of our progress up until the point of writing the interim report. There are, however, a number of features we intend to add before the final submission.

First and foremost, we will be designing an interface for the code we've written so far, we've decided that Java's implemented GUI system should be satisfactory for this task. This interface would be built upon with each new feature added to the code. It will also feature useful buttons for accessibility such as returning a list of all student names.

One of the biggest planned feature of our code is the method of filling the remainder of a student's preference list. Our approach is to fill the empty slots with projects that are less desirable as to lower the odds of a clashing preferences between two students. We plan on creating a hashtable linking preferences (key) to their popularity (value). This should allow us to easily distribute the least desirable projects and allow us to more easily design an algorithm to find the solution of best fitness.

We also intend to include a method that takes in a project and gives a list of each student with that project listed as a preference. This will allow us to see if particular students are only taking up the most desirable projects which would increase the odds of clashes and lower the overall fitness.

Another planned feature is the ability to print the final mapping of students to projects, allowing the user to see clearly what has been assigned to who when the best energy has been found. Finally, we aim to include the option to read in data from a different file as long as it is also tab separated, this will give the user the ability to update the list of students easily if there are any changes made to it later on.

Detailed Design

Before we started working on this assignment we first combined all of our individual code. This code contained four classes: *PreferenceTable*, *StudentEntry*, *CandidateAssignment* and *CandidateSolution*.
PreferenceTable takes information about a student from a tab separated file and stores it in a vector using the private method *loadContentFromFile*. This method opens the file and stores each piece of information as tokens. It is called in the constructor of this class.

PreferenceTable also contains a hashtable that holds a student's name and a *StudentEntry* object. Two methods are used to deal with the hashtable: *addToHash* and *getHashTable*. *addToHash* adds each student individually and is called later in *CandidateSolution* to fill up the hashtable. *getHashTable*, as the name suggests, returns the hashtable.

PreferenceTable also contains methods that randomly returns a student and a project. *getRandomStudent* deals with the first of these. This method gets all of the students from the vector and stores them in a brand new vector. It then gets a random entry of this new vector and returns that student. The next method, *getRandomPreference*, calls *getRandomStudent* to get a random student and then randomly selects one of their projects which is later returned. However, we do not want to get a preassigned project here so there is an *if* statement to deal with this. It simply looks for a new project is a preassigned one is found.

The final function of *PreferenceTable* is to fill a student's preference list with random projects. This method goes through each student and will fill their preference list with random projects using *getRandomPreference* but only if they have less than ten projects and they don't have a preassigned project organised. The method *hasPreassigned* is used to determine whether or not they have organised a preassigned project. This preassigned method is also used in the above method and is located in *StudentEntry.*

The *StudentEntry* class deals with each student as an object. It will be passed a student's name when it is being created. It will store their list of projects in a vector, whether or not they have a preassigned project in a boolean and the original number of projects they had listed. It also has a few methods to deal with this information.

One such method is *preassignProject.* This method will be passed a project name and check if a student has that specific project preassigned. The method *hasPreference* is similar but it will look for a non-preassigned project.

There is also a method, *addProject,* which will, when passed a project title, add it to that student's list of preferences. There is a check before adding a project to make sure a student has no more than ten projects and that they don't have a preassigned project. This will stop it from adding too many projects.

getRandomPreference returns a random preference from that student object. It uses a random number generator to get a project from the list stored in the class. Unlike the *getRandomPreference* in *PreferenceTable* this one isn't entirely random because, as mentioned above, it works on a specific student.

The final method of this class works with the vector of student projects. *getRanking* takes in a project name and will return its ranking. Its ranking is its position in the list i.e. their first preference will return ranking 1. If the vector does not contain the project it will return -1.

The methods in *CandidateAssignment* deal with randomly assigning a project to a student, the energy of that pairing and it also contains an undo method. The energy of a pairing is calculated using the formula energy = (ranking+1)*(ranking+1) and is returned using *getEnergy*. The ranking is retrieved from *getRanking* above*.* The aim of this assignment is to get the lowest possible energy for every student. The energy is used and expanded in *CandidateSolution*.

As previously mentioned, this class has an undo method, *undoChange*. When the student is assigned a random project their previous project is stored in the class. If they did not already have an assigned project then their "previous" assignment will be an empty string. *undoChange* basically sets the student's current project to the one they had before this random assignment that was stored above.

The constructor of this class calls the method *randomiseAssignment*. This method does the job of actually randomly assigning the student their project, as its name would suggest. The "previous assignment" used in *undoChange* is set here before a new project is assigned. To give a student their random project *getRandomPreference* from *StudentEntry* is used.

The final class of this base code is *CandidateSolution.* The constructor populates the hashtable created in *StudentEntry* and assigns every student with a random project. This list of student to project mappings is stored in a vector.

getEnergy will get the overall energy for all of the student to project groupings. It will do this by adding the energy of each individual mapping and any penalties. The penalties will add 1000 on to the energy if two students are assigned the same project.

The final function of this class is to calculate the fitness of these mappings. The fitness is just the inverse of the overall energy calculated above so the formula Fitness = -1*energy is used.

The remaining features are all in planning, none have been implemented yet. Some may get their own classes but others will be added to classes already present.

The first function we want to implement is to make sure every student's profile is valid, i.e. they haven't listed a project more than once, if they say they have a preassigned that they only listed one project, etc. This will be done in *StudentEntry* and *PreferenceTable*. A check will be added when adding each project to the vector in *PreferenceTable* to make sure it is not already present in the original list. Another check will have to be added in *addProject* to make sure the project being added is not in the list.

Making sure that a student who has said they have a preassigned project only lists a single project will be added to *StudentEntry*. This will be done by adding a method to check that if the student has indicated that they do indeed have a preassigned project that the vector containing the preference list only has one entry. This method will then remove the extra projects if necessary.

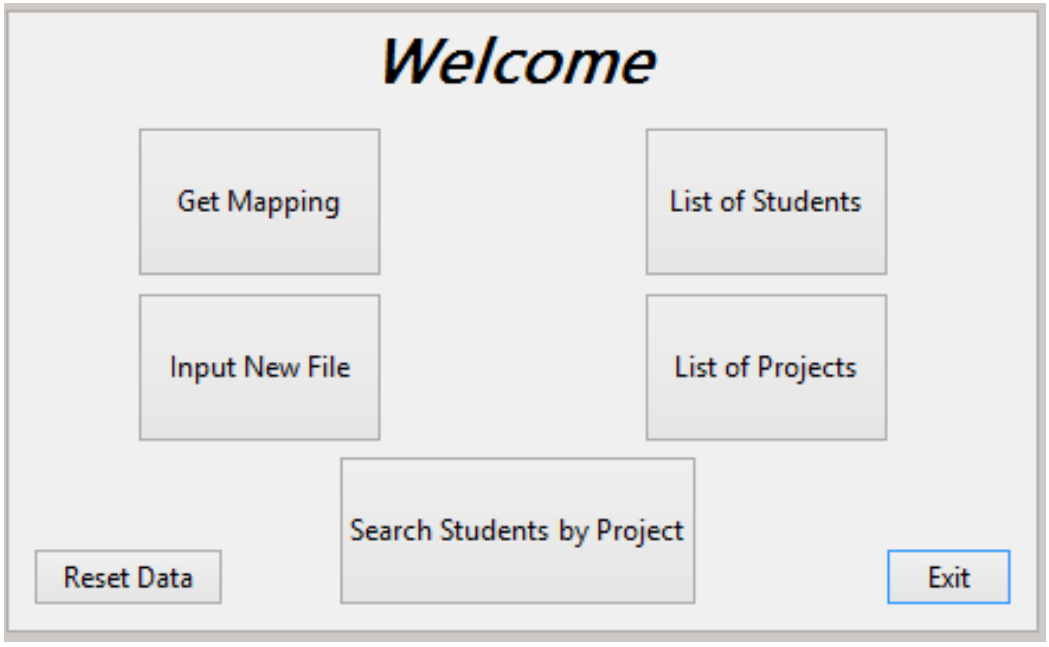
The next thing we want to do is improve how the code adds projects to student's preference list if they've less than ten projects. We want to do this to help improve the energy of the mappings. We want to avoid adding preferences that are other people's first choices, giving them a better chance of being assigned it. This function will be added to *PreferenceTable* where all the students' preferences are filled. A method will be created to handle this. It will create a list containing only first preference projects, excluding preassigned projects. It will then create another list that will be created by comparing the complete list of projects to the list of first preferences, only adding those that are found in one but not the other. This list of all the projects will be filled in the constructor when we're adding them for each student. We will have to take care not to add duplicate projects here as well.

Another method will be created to find out which projects are the least popular. These projects would be ideal to add to incomplete preferences as it will make finding the ideal energy easier. To do this we will need to create a hashtable that will store each project and how many times it was chosen by students. This hashtable will be made in the constructor when adding the projects. When a duplicate is encountered it will increment the number of times it was chosen. The method will take this hashtable and sort it with the least popular projects at the top and then ascending from there.

Then, when adding project, we will try and use these two new lists to find out which projects are the best to add. This will, hopefully, help us when searching for ideal energy and fitness levels.

Another part we want to add to the code is to disallow duplicate allocations. To do this we will add to *CandidateSolution*. When adding each mapping to the vector in this class we will check the previous entries to make sure the project being assigned has not already been assigned. If it has already been assigned *CandidateAssignment* will be called again for that student to get them a new mapping. This will also allow us to remove the penalty resulting in a lower and better energy.

The next function that will be added will be a user interface. We will construct a GUI that will be able to implement all of the important methods to the user.



Sample menu

The Get Mapping button will get the final mapping and its energy. The List of Students button will bring the user to a new menu which will give than an option of three different lists to generate: One containing students with preassigned projects, one consisting of students that do not have a project preassigned and, finally, a complete list of every student. The Input New File button will allow the user to enter a new file rather than use the one already hardcoded in. List of Projects will print a list of the projects sorted by popularity. The Search Student by Project option will allow the user to enter a project name and return a list of the students that wanted that project. Finally, there will be Reset Data and Exit buttons which are self-explanatory.

Finally, we will need to create an algorithm to produce the ideal mapping. This mapping will have the lowest energy level. Some of the functions listed above will help shorten this algorithm to an extent as they would ideally have filled each student's preference list with projects that would not end up negatively impacting another student's energy. When planning the algorithm we will be looking at two stochastic techniques as discussed in class, Simulated Annealing and a Genetic Algorithm.

Schedule and Work Breakdown

After we brainstormed and came up with the ideas mentioned above, our next task was to distribute the work evenly between the members and assign achievable deadlines to each person for each task. We first made a table listing all the jobs, breaking larger jobs up into smaller tasks, and decided how many people would be required to complete each task. Everyone then had the opportunity to voice which jobs they would like to do and were capable of doing.

The first job on the list was reading through everybody's programs and deciding whose code we would use as a starting point. This was a four person job as everyone needed the opportunity to voice why we should use their code and to bring to the group's attention any extra features their code had that others may not. It was obvious this tasks had to be completed first as it would be impossible to continue if each member was working on a different program. We assigned the deadline to be the 11th of March so that when we returned from midterm break we could immediately start programming.

The next job we discussed was insuring that every student conformed to the rules i.e. that each student listed 10 projects, did not list the same project more than once, etc. We felt this was a relatively short job and that it could be completed by a single person. Mark volunteered for the position. Since we needed correct information from the students before continuing this job must be completed promptly so we decided to set a close deadline of the 30th of March.

From this idea we derived another we thought would improve satisfaction down the line. The idea was if a student listed less than 10 projects that our program would fill up their preference list with the least popular projects. This idea was more complicated than the previous and we felt it required two people to complete. Niamh and Karl believe that would work very well together as their ways of thinking are slightly different and therefore complement each other. It is because of this Niamh and Karl offered to on this two person task. The deadline was set to the 13th of April as we thought this task may take a bit of time.

Another idea to help improve the overall satisfaction of the solution was to prevent assigning a project to a student if that project had already been assigned to another student. This did not strike us as an overly complicated task and therefore we assigned this to be a one person job, Caroline agreed to be that person because she was the only person who had yet to be assigned a job. As it did not seem overly complicated it should not require too much time and thus the latest date assigned for its completion was the 4th of April.

The next task on the list was coming up with the optimum algorithm with which to calculate the overall satisfaction of the solution. We believed that in order to create the best algorithm we needed four different mind-sets to come together and offer their solutions, discuss the merits of each idea and decide which idea was best or if there was a way to combine our ideas to create the most efficient solution. So it was decided all four members would contribute to this task. The overall goal of the project is to find the mapping of students to projects with the best satisfaction due to this fact we believed this would be an ongoing job and gave it the latest deadline of the 25th of April.

Another idea we needed to discuss was the interface/menu. The interface will be one of the larger jobs we have to do, so in order to ensure everyone did a fair amount of work we decide to break this job up into smaller tasks so we could all work on it together. Before adding functionality we first had to make a basic interface which had all the buttons our final product would have but not all of which worked. Each task completed would then render a button functional. We thought that making the skeletal interface would be a two person job and so it was assigned to Niamh and Caroline. Niamh had previous experience with GUIs and we would use that knowledge to our advantage but Niamh could not work alone so she chose to work with Caroline as they also work well together. We also wanted to ensure that for the two person jobs it was not the same two people working together all the time.

We felt the basic interface should do four things and as there were two of us working on it, the first was to allow the user to enter a new file to be used, of to continue with the build in file, the second was to have a reset button on the main page in order to return to the add file page. Caroline volunteered to do these tasks. The third and fourth tasks were to allow the user to enter a student name and have a list of that student's preferences returned, and to have an exit button that would exit the program. Niamh offered to complete these tasks. The deadline for the completion of the interface is the 25th of April we need the basic interface to be finished before this and set it a deadline of the 15th of April. We chose the 25th as the final deadline for the interface to allow enough time for integration and vigorous testing of the code as a whole.

The rest of the tasks were to add functionality to the remaining buttons on the interface. The next feature we wanted our menu to have was the option to print out a list of students. Mark was already working on student validity, identifying which students had preassigned projects and which didn't so he thought he would easily be able to use this information to his advantage and would easily be able to form lists to be printed. The deadline of the 18th of April was set.

The next function we wanted our interface to have was the allowing the user to enter a range of satisfaction. Again, we thought that one person would be sufficient to do this, Mark was also assigned this job. We thought it would be best if both interface related tasks Mark was working on had the same deadline and so April 18th was set as the date of completion for this task also. Mark will also have to create back buttons that will navigate the user back to the main menu. These are to be done as the as soon as possible after Mark's other tasks but the absolute deadline was the 25th when the interface was to be complete.

The next button to be programmed was the one to print a list of the projects. As Karl had been assigned the tasking of filling students preferences with less popular projects he knew he would have to rank the projects by popularity anyway and therefore thought this task would be a by-product of his earlier task and it would make sense for Karl to be responsible for this task. As this task was closely linked to Karl's other task which was due on the 13th we set the deadline for this task to also be the 13th.

The final feature we wanted our interface to have was allowing the user to enter a project name and get back a list of the students that have listed this projects as a preference. Since Karl was dealing with project popularity already he thought this task would be closely linked to his other tasks and offered to work on this task. We decided that the 23rd of April would be a good deadline so Karl was not overloaded with work and it would not affect Karl's earlier deadline of the 13th. Karl would also have to implement back buttons for this functions. The deadline will also be the 25th of April.

The final two jobs to be completed were a matter of housekeeping. The first of these was testing. We thought that we should all be responsible for testing both the code we wrote and the code we didn't. It is important to test the code you write as you write it but it may also be useful for someone who didn't write it to look at it with fresh eyes as they may see something the programmer missed. It is because of this that code will be tested in groups of two. Groups will change depending on what is being tested. One tester will be the programmer, the other will be a rotation of the other three members, again, to avoid a pairing of members and a division in the team. Testing will be an ongoing job but the testing for each task should be completed within a week of the task being finished. All of the code must be completed by the 25th to allow time for testing the finished product as a whole. Therefore the deadline for all testing to be done is the 29th.

The final job to be done is one we have entitled “code janitor”. The code janitors will review all the code written, as there will be four people with four different styles contributing to the one project the code janitors will ensure the code is consistent and easily read and understood. The job is to ensure that the code is as neat as possible, that variable names are explanatory, code is efficient, no commented out code is left in etc. The job of the code janitors is very important as the easier the code is to understand the easier it is to extend. Similar to the code testers, cleaning the code will be done in pairs comprising of the programmer who has extensive knowledge of the code and another member to give a fresh perspective. Again, like above, the pairs will be varied as much as possible. The cleaning of code should be done as soon as possible after the task is complete and will therefore have periodic deadlines with the absolute deadline being the 29th of April. Code janitors and code testers will be assigned regularly based on the workload each member has.

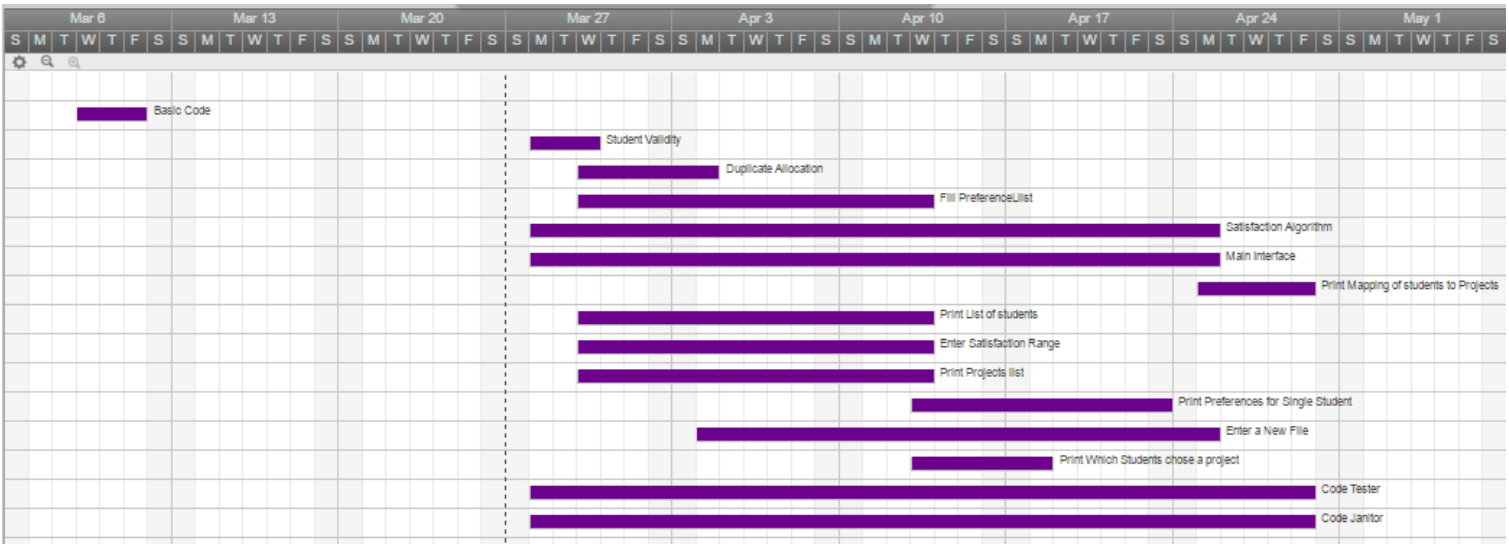
Once all the jobs were assigned and the schedule table was complete we reviewed the allocation of jobs ensuring that no members were assigned too much or too little work. First, we looked at the interface. The interface was broken up into eight tasks and each member was assigned two tasks. There were then two jobs that all four members would contribute to, making the base code and designing the satisfaction algorithm. Code Janitor and Code Tester were two jobs that would change ownership over time but that also require equal contribution from all four members. There were then 3 jobs left, one larger, two smaller. Caroline and Mark were assigned the smaller jobs and Niamh and Karl were to do the larger job. We were all satisfied that the overall workload was evenly split.

This table was the result of our plans:

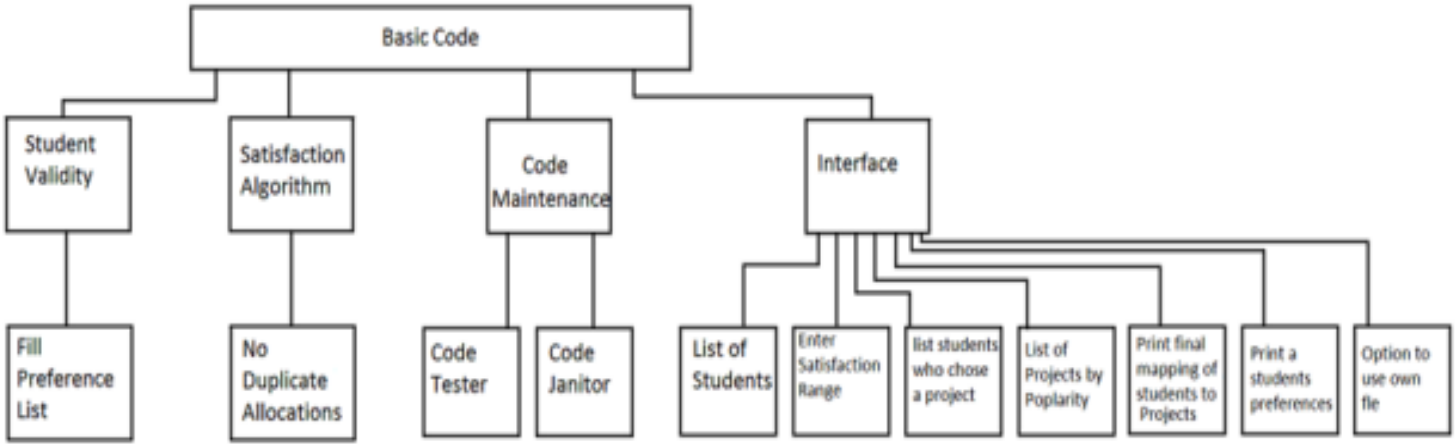
Job	**Number of People**	**People**	**Deadline**
Making Code	4	All	11th March
Student Validity	1	Mark	30th March
Fill projects	2	Karl & Niamh	13th April
No duplicate allocations	1	Caroline	4th April
Code Janitor	2	Mix group weekly	Weekly deadline
Code Tester	2	Mix group weekly	29th April
Satisfaction Algorithm	4	All	25th April
Interface Features			
Main Interface	4	Niamh, Caroline	15th & 25th April
List of students	1	Mark	18th April
Enter range of satisfaction	1	Mark	18th April
Student by project	1	Karl	23th
List of projects by popularity		Karl	13th April
Print final mapping	1	Caroline	25th
Student preferences	1	Niamh	18th
Option to upload new file	1	Caroline	25th
Exit and return buttons	2 + 2	All	25th

Finally we reviewed the deadlines we set for each task. We were happy that we had allocated an appropriate amount of time to each task, keeping in mind that each person will have more than one task to do, some of which will be done alongside another task and some of which will start after another task is complete. We used a Gantt chart to see what jobs could be done independently and which jobs relied on others. We also used a Work Breakdown Schedule to examine the amount of work involved in each task. This helped us visualise the deadlines we had set and helped us determine if we had allowed sufficient time for each task.

This was the chart we used:



This is the work breakdown schedule which we created:



Reporting Structure

One of the first things that we had to decide was how the communication was going to be maintained within the group. We also had to determine how we were going to measure the progress of each individual in terms of their commitment and the workload in which they are assigned. We decided in our first group meeting, before the midterm break, how we were going to accomplish this. Various ideas were put forward, both ones that were accepted and rejected. In the end we decided on a few different methods of communication for this project.

In relation to the code we all agreed that the easiest way to maintain, add and update the code would be to use Github. It is a service that all the members in our group are both familiar and comfortable with, as we have used it in previous assignments. Github also offers us another service: every time a member of the group updates the code then it will be recorded by Github. This will allow us to keep track of everyone's progress. Github tracks everyone's commits by creating bar charts for each individual. These charts will help us make sure that everyone in the group puts in equal effort. Github will also make sure that each individual completes the parts of the assignment that had been assigned to them as we will be able to see who has updated the code and when.

The second form of communication that we decided to use was a Skype group. Since each member of our group lives in a different county and goes home some, if not most, weekends we decided that a Skype group would make it relatively easy to discuss issues, ask questions or ask for advice. It also allows each of us to ask the other members of the group if they are up to date with their work and if they are on track in terms of deadlines. Using Skype allows us to use the function of screen sharing. In the case of an error or problem in one of our tasks it will allow the other members of the group to look at the code and offer their assistance. The use of Skype at the weekends will also allow us to plan each of our weekly meetings. It allowed us to discuss what we wanted to accomplish over the two weeks off as well.

Because we had to write this report over the midterm we decided that, instead of attempting to use individual word documents that would be constantly overwritten by exchanging work, a private Google document would be the easiest way for the group to write it. Using the Google document also allowed each member to keep up to date with the latest version of the report. We assigned each person their portion of the report in a group meeting. To maintain individual effort and to monitor progress when writing the report we decided that each person would take a different heading from the notes. This would ensure that every member of the group had an equal amount of work to do. Once the report is complete the use of the Google document will allow everyone to proofread each other's work. This will allow each member to share their ideas and opinions on what has been written. The use of a Google document was a very efficient way of keeping in contact in regards to the report and, after the completion of this report, it will also be a very useful way to note any ideas that we might have. It will also help us keep a record of what we have done and what we have yet to do.

The final method agreed on to keep up communication was a Facebook group. We decided that out of all of the ways to keep in contact a Facebook group would be the best. Facebook is a platform that each member of our team has access to both on our laptops and phones. This allows convenient communication between every member. It will allow us to create polls that can be used to determine when the weekly group meetings will take place too. It will also allow us to post helpful links and resources. Another advantage of using a Facebook group was the notification system that it has. Every time someone posts in the Facebook group each member will get a notification. This will make every member of the team aware of any links, problems or queries posted almost immediately.

Along with the decided methods of communication we also rejected a few. One of the methods that was rejected was the use of a blog. The reason that a blog was rejected is that we didn’t have any experience of using a blog and didn’t know how the notifications would work, unlike the Facebook group that was discussed above.