Project Report

SOFT20091: Software Design & Implementation

****

**Project Title:** Bluebrook Airport

**Members:**

Samuel Crane – N0626631

Paul Havelin – N0623631

Adil Rajal Hussain – N0626369

Joe Zalewski – N0623403

CONTENTS PAGE

1. Overview

For this software implementation project, we decided to look at a system used by a small domestic airport that deals with arrivals and departures, seat allocations, emergency flight plans and other various features. The aim of the app will be to streamline and provide accurate, important information for flight administrators to ensure that emergencies are dealt with efficiently and safely and the day-to-day running of the airport runs smoothly.

1.1. Functional Requirements

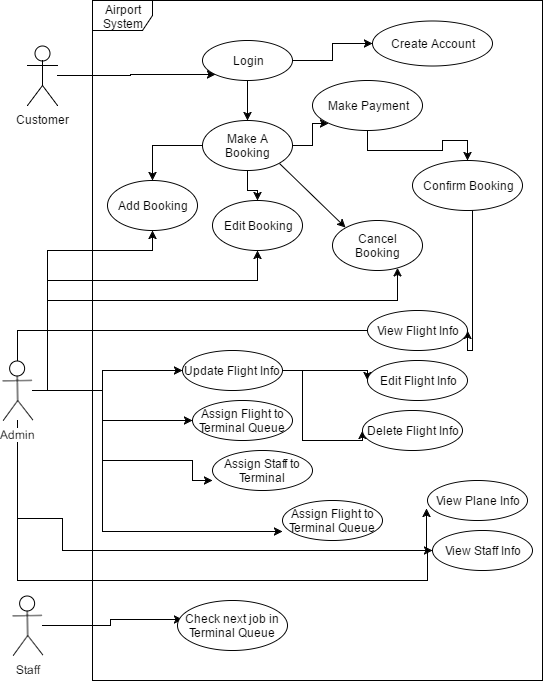
* Manage all flights
* Store airplane related data including passenger’s data, plane capacity, which seats are booked.
* Monitor departure and arrivals. These two types of information should be displayed in different monitors across the screens available in the airport.
* Provide search options: Allow users to search for flight or passenger info, depending of their level of access.
* Display seat allocations (if GUI display booked seats in red – user can’t click on them and display non booked seats in green?)
* Handle security: check passengers identify, if ok let them pass through the airport
* Passengers/Security and admin have different level of access to the system

|  |  |  |  |
| --- | --- | --- | --- |
| *FR* | *Requirement* | *Implications* | *Type of Requirement* |
| FR1 | Manage Flight | Data Structure required for handling:  -Schedule  Live Departure/Arrival  -Flight information (number, no of seats, airline etc.) | Functional |
| FR2 | Manage airport spaces & staff | Data structures required for handling:  -airport runways  -airport bays  ground force  -assign staff to bay for landing airplane | Functional |
| FR3 | Manage passenger info | Edit passenger personal details (name, address, booking history etc.) | Functional |
| FR4 | Manage Staff info | Edit staff personal information (name, address etc.)  Assign staff to terminal | Functional |
| FR5 | Assign physical spaces to flights | Correlate requirements 1 and 2. Generate error messages if there is an attempt to assign a runway/airport bay being used in a particular time slot.  Prioritise landings and departures. | Functional |
| FR6 | Login system | Create account for new user or staff member  Log in to system using stored credentials | Functional |
| FR7 | Book/Cancel/Edit flight | Staff and passenger can create, confirm, edit, and cancel booking. They can change seat and class. Passengers can also make payment | Functional |

2. Requirements In-depth

2.1. UML Use Case Diagram

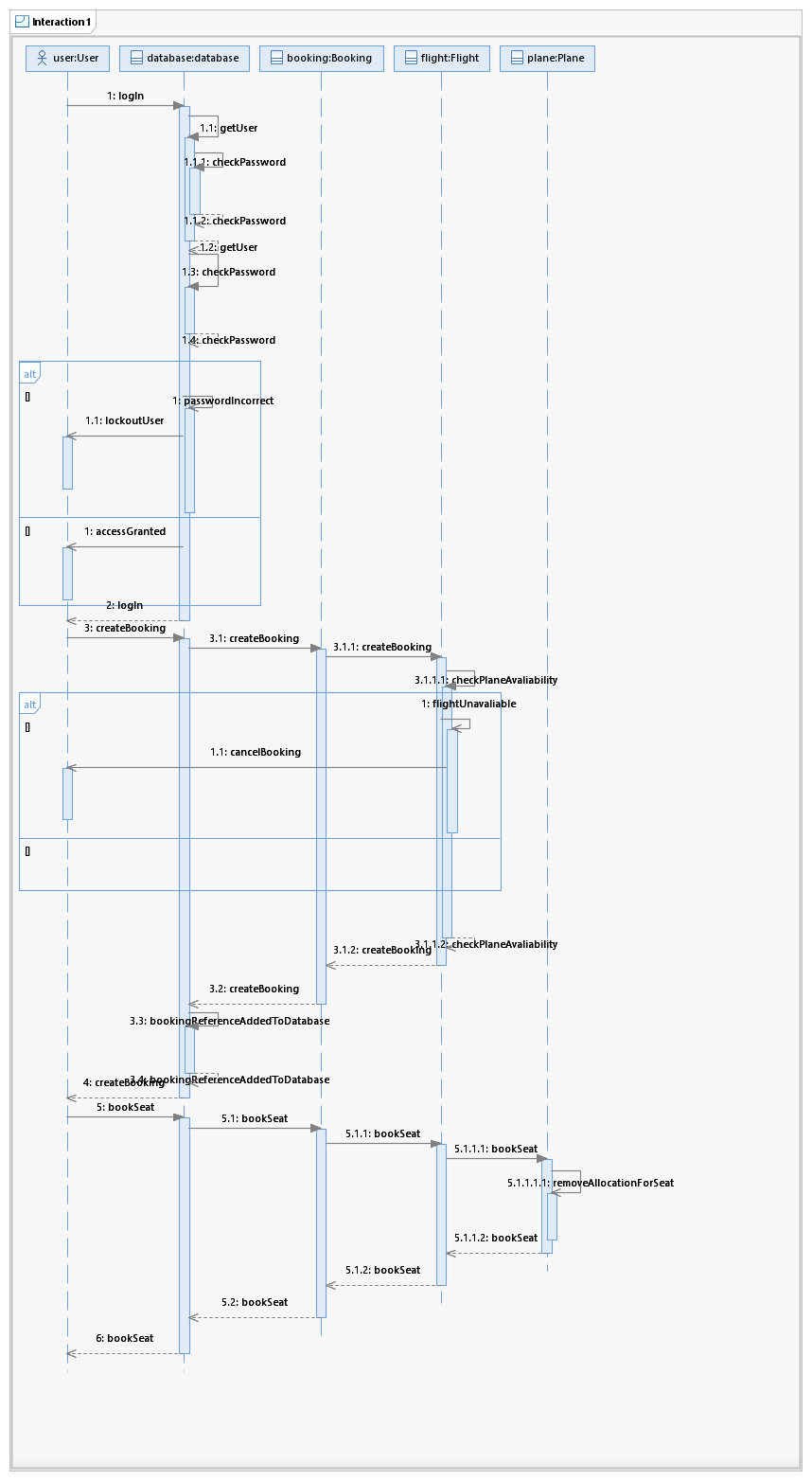
By designing a use case diagram the system is able to be visualised and to see how interactions will be taking place between different elements. For example, you can see how the Administrator has access to very high level functions that the Customer will also be able to see once completing steps beforehand.



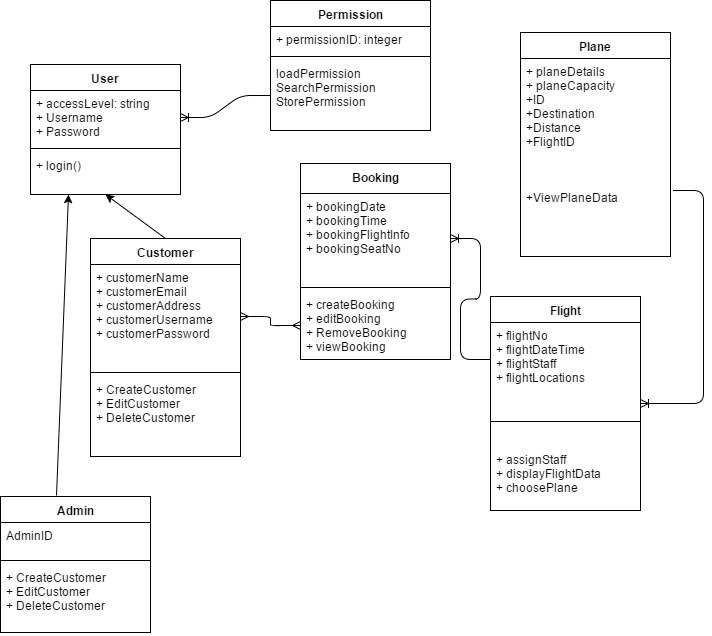
More in-depth, text-based use cases to describe this diagram can be found in appendices.

2.2. UML Activity/Sequence Diagrams

A sequence diagram is useful for seeing how different functions will be passing through specific elements and visualising what kind of structure functions will have to be as we will see what kind of variables will be passing through and how each element will have to communicate. This is essential during the designing code process simply because of the fact code can be structured by looking at this diagram and provides a significant starting point.



2.3. UML Class Diagrams



3. Approach

3.1. Chosen Methodology

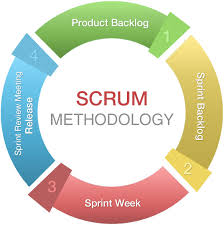
Methodologies are methods which are used to carry out projects and manage development teams in the IT industry. Many different mythologies fall under the agile mythology category. The main difference between agile methodologies and more traditional mythologies (Waterfall for example). Unlike waterfall model, agile mythologies follow a sequential approach which gives us room to change the requirements of the projects if we were to decide to aim for higher grades during the project.

This also gives us the chance to work on our project and start the very basic features without spending too much of our given time doing extensive planning, instead we can start building on the early features of the project.

This also allows us to start testing the project as we work on it, as opposed to only leaving the testing as one of the last phases, so we can make adjustments on the early features before we start working on the more advanced features as opposed to testing the whole project at once, which may lead to us fixing a large amount of errors during the final weeks of the deadline.

One of the disadvantages of agile methodology is that if it not followed consistently, the project can become stages of code sprints, which could lead us to having an incomplete final project; but the reason we are choosing an agile methodology over waterfall is, if we do decide to go for the higher grades; we can adjust the requirements and we can add in new features once we have the basic features complete.

More specifically, we are following the Scrum agile mythology. This method is more beneficial to us than other methods is because of the constant meetings we have, so we can stay up to date on what has been complete, and what still needs to be worked on. This ensures we stay on top of the project work as opposed to delaying the work and completing other modules, because we won’t have to present our work at weekly meetings if we were using waterfall methodology for example.



3.2. Description of Agile (Scrum)

*Planning/Analysis Phase:*

In the very first phase of the lifecycle, our team is going to be planning and analysing extensively so we can start to work on the basics of the program as soon as the very first sprint. In this phase we plan, what our software will do, what features it should or shouldn’t have.

*Sprint Planning/Meeting:*

In this stage of the lifecycle, we assign work to the members of the team; determining what work which will be completed by each member. This will determine what work they will cover in the sprint week and present during the next meeting. Every meeting apart from the very first, we will discuss what we have completed since the last meeting, the problems we encountered and if we fixed them. Work is then assigned for the sprint.

*Sprint:*

During this phase, which is the key phase of the lifecycle, during this week; work that was assigned to the members is completed. In this phase we plan on actually creating our software, creating the actual functionalities, working on the documentation and testing the program. Each sprint will be one week, we will then return to the last phase of our lifecycle, iterating the meeting and sprint phase until the project is complete.

System Release:

We will have multiple releases for our software, the later versions will have more features and will hopefully work as intended and early features may have bugs that will need to be fixed. The final release of the software is the one we will upload as the project submission and demo to our lab tutor. As it will be the most refined and polished version of the project.

3.3. MosCow Anlaysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Must Haves** | **Should Haves** | **Could Haves** | **Won’t Haves** |
| Login System | Book specific Seats | Landing areas available for planes | Change plane route |
| Book Flights | View plane information (Engine size etc.) | Staff information on plane. | Delete/edit plane information (Engine Size) |
| Edit Bookings | Automatic updating flight departure board | Payment system for customer | 3D visualisation |
| View Bookings | Arrival times of planes | Admins change flight information |  |
| Edit Plane information | Two different type of users (Customer and admin) |  |  |

3.4. Object Orientated Programming

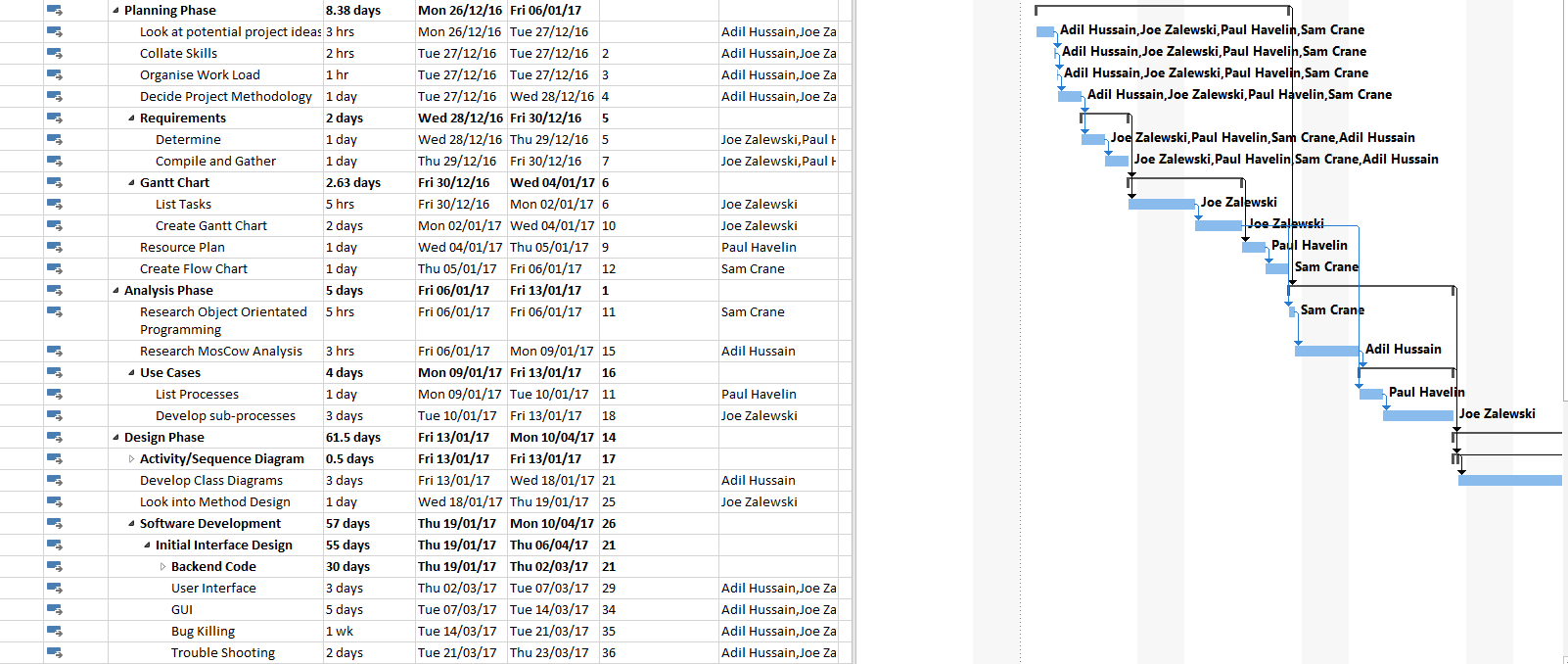
The Bluebrook Airport system utilises an object-oriented approach in its code design. The entire concept of the program revolves around objects interacting with one another, such as users, flights and planes, making object-orientated programming ideal and efficient. Additionally, in a system like an airport booking system, maintainability is something that should always be thought about as the travel industry evolves and object-oriented design lends itself very well to this – adding functionality to objects is a much easier process than going through a procedural based program.

One useful feature of object-oriented programming is inheritance, which is something that has been utilised with the User class. There are several methods that all users should be able to perform, such as update their details, but additionally each different type of user will have different methods that they want to invoke. For example, regular accounts exclusively would want to view their flight details and their frequent flier points whereas staff accounts would want to be able to print off flight chart details and manually override flight details if required. None of these things should be accessed by the other class, which means we can make use of inheritance to make two subclasses, UserRegular and UserStaff. This is flexible and helps with maintenance, as we can make specific user subclasses for each staff role to allow them to do perform their own methods without having to rewrite all the shared methods inherited from the User class individually.

Another feature of an object-oriented system is that allows for easy maintainability is that it is much easier to model the system through UML (Unified Modelling Language). This not only has benefits in the design process and initial building of a system, but a well-designed and maintained model makes it a lot easier to sustain and further develop already existing systems because of how easily read these models are and how clearly they show all the relations between classes.

3.5. SDLC

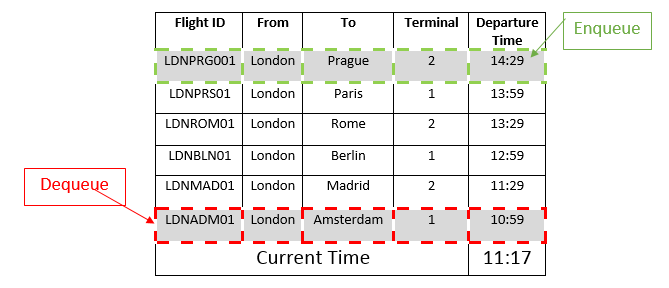
The software development life cycle was created using Microsoft Project in the form of a Gantt chart. The team decided that this was the best option as the software not only allows the visualisation of how the project should be coming along at all stages but also has resource allocation and management that gives an accurate and realistic view of how the project will use assets.



The complete version of this Gantt chart can be found in the appendices.

3.6. Data Structure

The chosen data structure for the Bluebrook Airport System is a linear structure; a queue. The purpose of a queue is to organise data is a First In, First Out (FIFO) structure. This is implemented in the flight time board, which will display the current system time, the flight ID, flight locations, terminal number, departure time and current system time. There is a fixed amount of flights displayed on screen at one time, previously sorted in the system by departure time. The system/current time exceeds the time on the lowest sorted time, the time dequeues from the timeboard and the row is hidden from the time board, as this happens a later time, previously not displayed will be enqueued on the time board. The process loops throughout the day until all flights for that day have departed the airport.



**Advantages:**

**Speed** – queue are quick to implement, use and maintain as there is only two main stages to organising the data (enqueuer/dequeuer).  
  
**Memory** - as a queue data structure is linear, it will take up more memory than a non-linear structure.

**Disadvantages:**

**Sorting** – Before the data can be put in the queue, it must be sorted first, in the case of the Bluebrook airport system, data must be sorted by time closest to system time.  
   
**Decision**:  
The decision to use a queue data structure was an obvious choice, as is used in real time board systems as it is the most accurate, easy to understand method of displaying sorted time data closest to a certain time (current system time).

3.7. Algorithms

For the implementation side of this module, the task was to implement two different types of algorithm that are efficiently and appropriately used to make the project more streamlined and relevant to normal coding standards, especially when discussing the size of the system.

3.7.1. Searching Algorithm

For the implementation of a searching algorithm into the Bluebrook Airport system, the chosen algorithm is a Binary Search Tree. The purpose of using a searching algorithm in the system is to search for data within the system. Data can include flight data, plane data and customer data, staff rotas.  
  
Each vertex has each 0, 1 or 2 branches, which holds a record/key. At each vertex, smaller keys than the vertex are added to a subtree on the left, larger or equal keys are added to the subtree on the right.

Main operations are:

Search

Insert

Pre-order Traversal

In-order Traversal

Post-order Traversal

Steps included (in the example of a flight time search):

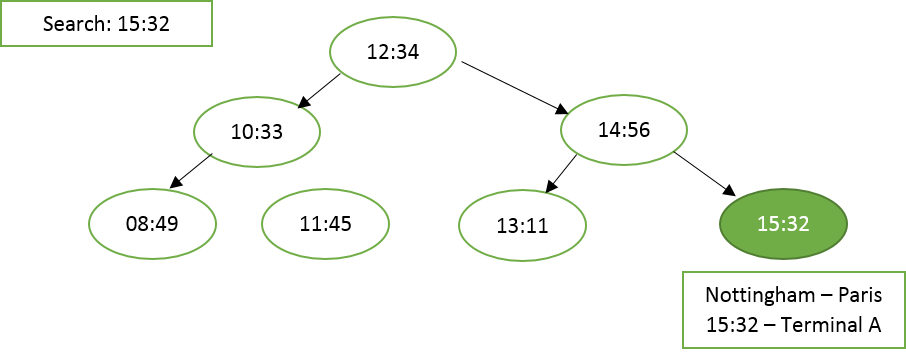
User enters a flight time

System uses the first flight time element in the xml file, this time becomes the root node

The system cycles through the next flight time, if it is less than the root, the time is added to the left side of the tree

If the time is greater or equal to the previous flight time, add to the right side of the node.

An example can be seen below in this diagram:



Advantages of this is that it is simple to implement and can be used for a variety of different data searches, for example flight, customers and staff. Despite this, data must be sorted first in order for this to work effectively.

3.7.2. Sorting Algorithm

For the implementation of a sorting algorithm into the Bluebrook Airport system, the chosen algorithm is Quicksort. The purpose of using a sorting algorithm in the system is to sort all flight times in order of earliest, this is essential for the time board as all flight times are added to the queue data structure for the time board.

Process of Quicksort in the flight time board system:

The system will select a pivot point from the unsorted list of flight times from the XML file.

Place all flight times earlier/later than the pivot flight time.

Sort sub list of flight times less than/greater than the pivot flight time.

Ignore sub lists if flight times is empty.

Place all sorted flight times in the time board queue

Amount of comparisons: n²/2  
Amount of flight time swaps: 2 N log N  
Big O’ Notation: O(N log N) – Linear logarithmic (Increasing growth with N for times)

An advantage of using Quicksort is it has the fastest computational speeds of the other sorting algorithms, however a downside would be that it is difficult to implement in comparison to other algorithms.

4. Implementation

4.1. Software Testing Process

[TO ADD]

4.2. Patterns

The purpose of patterns in the Bluebrook Airport system is to make the programming easier to understand, fix bugs, ensure future errors.  
  
4.2.1. Separating Methods   
--------------------------------------------------------------------------------------------------------------------------  
Name: Make Booking  
Description: The user will view empty seats, choose a seat then save their booking in XML  
Functions used: AddBooking / GetCustomerInfo  
--------------------------------------------------------------------------------------------------------------------------

To separate the various areas of the system, commented sections are used. Commented sections are useful as they help identify the purpose of each area to ensure the user can easily find and understand each section in a simple to understand manner before viewing the functions and variables. An example is displayed above for making a booking, giving the section a name and brief description of what occurs in this section to help other users easily understand, where appropriate, the commented section includes which functions are used in this section, this can help identify bugs easily and prevent future errors.

4.2.2. Variable Names

|  |  |  |
| --- | --- | --- |
| **Customer** | **Booking** | **Plane** |
| cFirstName cLastName cPhoneNo cEmail | bBookingNo bSeats bFlightNo bFlightName | pCapacity pStatus pMileage pTerminal |

For the variable names, the declarations are split into sections using a lowercase identifier such as in the example above such as a lower case ‘c’ for customer variables, a lower case ‘b’ for booking variables. The purpose of this is to be able to identify where variables come from when they are called in later function to reduce ambiguity such as if the customer variables are used in the booking section to link a customer to a booking. These variable names ensure any user can understand the program easily and reduce errors in future as the user can identify the origin of a bug easier.

4.2.3. Function Names/Declaration  
For the naming of functions, nouns are used to clearly distinguish a function from variables or class names. Comments are not needed, as the functions should be descriptive of the purpose of function linked with the commented section, in which the function is defined and used.  
  
**Public**   
The purpose of using functions within public classes is to ensure any class has access to the function such as viewing flight info, which can be done by all types of users e.g. customer, staff or admin.  
  
**Private**  
The purpose of using functions within private classes is to add security to the program, in which only the current private class will have access to a specific function such as admin functions cannot be used by staff or regular customers.  
  
**Protected**  
The purpose of using functions within protected classes is to ensure only the current class and subclasses have access to function, such as different levels of staff.  
  
**void Functions – Verbs (an action)**  
void functions are used throughout the Bluebrook Airport, they are unique functions which do not return a value, they are defined as camelCase using verbs as they are performing an action within the system such as the ‘createXML’ function, which is used create an xml file for the customer data to be stored in.

**Pre-Processor directives – ALL CAPS**  
Pre-processor directives are either defined in the C++ properites for the project or using a #define at the top of the program using #define followed by the directives in capitals. An example of this is the /////// directive in the /////// section of the system.

**Booleans**  
Boolean functions within the system are given correct Boolean names to return either a true or false output, an example for a function with the purpose of searching an xml file for a flight within the system is named ‘doesFlightExist’ as the function will return a value of True if the flight is found or a value of False if the flight cannot be found in the system. The purpose of this name is to include

5. Results

5.1. Testing Approach Reflection

[TO ADD]

5.2. Reflection on Performance

The evaluation of the performance of the project is paramount due to being able to look at how the project can be added to, maintained and also how it runs if it were an actual program being used. This is done by looking how fast it compiles and how easy it is to be taken elsewhere and still ran and how much memory it takes up when loading.

5.2.1. Reliability

The fact that the system will always complete its required job regardless proves that it is reliable, however, as stated later on in this evaluation, it does require the specified files for the program to work properly as it doesn’t create its own XML files. However, as the code is modular in its programming style, it can edited and still work in full, for when a part of the program is under construction.

5.2.2. Security

The security of the program is easily compromised. It holds user data as static strings in XML files that can be easily opened and viewed using file viewers, including passwords and email combinations. If this was to become a release it would have to be hashed and then decrypted when passwords are entered. This is also an issue due to the fact sensitive plane data is also set up as static information so can also be opened, which could compromise the planes and also be a threat to the passengers safety.

5.2.3. Portability

The way the program is set up is it users various XML files that are scattered in the source directory, meaning that if this was moved around different machines, it would have to have all the files added to the same directory otherwise it wouldn’t input correctly. It wouldn’t let the user know, either, when a file cannot be found and would just stop working, which would take away from the user experience.

To improve this, we would have to add a directory that holds all the user information and the XML files, then this can moved around with the release rather than having to add everything later on. Or, another way of getting around this issue, is to be able to create the XML file if it doesn’t exist previously.

5.2.4. Maintainability

The program can be looked after constantly using the very easily viewable, split up functions and then code can be added and changed. Due to the way the menu is set up, when a menu item is being worked on it can simply be taken out of the menu and not change how the program functions. Obviously, it wouldn’t be able to be inputted either. This is a slow way of maintaining a program but it is effective, although basic.

One of the only big issues with maintaining the program is the large size of it, as it would need someone who knows what they’re doing to change the program, as someone who didn’t and didn’t research the program enough could cause fatal errors to the system.

5.2.5. Scalability

The way the code is set up allows the program to be easily added to later on in development, as each function is called from the menu and then the functions are kept separately. This also means that it can be synchronised when multiple people are adding to the code and Github commits can be completed more successfully. Also, due tot eh fact it is a command line based program, it is able to be added to slightly easier as new graphical interfaces don’t have to be made and instead one can just create a new function and add a text based number code that locates it in the menu.

Another aspect of how scalable the program is being the fact it can work with only a few flights and plane information and also millions of the same data. How efficient the program is could be tester however due to the data structure of SQL it would allow millions of data entries and still be as easy to read from, although it might take a little longer to search through the file. This would be necessary for the system due to the fact it’s an airport system that would contain lots of flight codes and passenger information.

6. Conclusion

6.1. Professional, Social, Ethical and Legal Aspects

Due to the information that is collected by the system, one must be able to protect their sensitive information efficiently. As of current, the files are simply string and static, which would have to be strengthened and encrypted if this were to be a real system being used for an airport. It would also have to be hosted on a server, which would allow it to be used even more inappropriately, requiring even stronger efforts to maintain the network.

Due to the collection of user information, we would have to look at the maintenance of privacy for the users of the system, whether they be customers, staff members or administrators. A program of this scale should comply with the Data Protection Act 1998 as personal information is being processed constantly to ensure that user information isn’t lost and isn’t misused by hackers and third parties who have gained unauthorised access.

6.2. Reflection of Methods

As a group we decided upon using Agile (Scrum) as our chosen methodology. After doing a lot of research as a group and discussing what we wanted to do as a project and how we code as a group, we found out this was the correct thing to do. By allowing us to release separate working products each time we had a ‘sprint’, this allowed us to look at how much work we had to do for the next release as defined by our project milestones which were tracked separately. By using this method, we could bugfix on each release and also have pre-defined releases, which gave us a good scope for the project and allowed us to work on priorities for the next release. Also, by using Agile as a general methodology, we could take a step back at various points throughout the lifecycle and change how we were working and what we wanted to change.

One of the drawbacks of using Scrum is the fact that it is constant code development with not much time in-between. As soon as one release was done, another release was being worked on with no much time to evaluate what we did well, what we didn’t do well and what we need to improve, which lead to confusion after releases on what has been achieved and who has contributed effectively. Due to this, a different methodology might’ve been the most appropriate for this kind of activity, one that allowed more reflection and evaluation.

By developing a Gantt chart that mapped out our software development life cycle alongside our chosen methodology we were able to see how timings were working and how far along the cycle we are and where we should be aiming for next. It also allowed us to simulate a real life project and resource map to take a look at how different people will be working on different things. It proved that we were terrifically over subscribed for the workload we were taking on and if this were a real life company there would be a team of at least 10 working on this at any one time (when taking into account working days, weekends, overtime and payment).

Developing a detailed flowchart was a good move as we were able to look at how different elements communicated with each other, in the same way we used the use cases. The flowchart accurately matches the final version product which is a huge benefit as it shows we had a structured development process for our project and also shows our chosen methodology, although being impractical at times, worked. The flowchart could’ve included slightly more detail to allow alternative flows, as set out in the use cases in appendices.

Deadlines and milestones were predefined at a very early point in the life cycle to give us an idea of how far we were getting, how much work we had left to do and where we were falling short. This allowed us to chase up issues that were coming up ASAP so that we didn’t fall behind on the deadlines and by having separate releases, inline with our methodology, allowed bug fixes and evaluation. Despite the positives, it was difficult to keep on track of the milestones at certain points and when members were falling behind on documentation work.

Github turned out to be the fifth group member during this process. Although there was issues with branches early on in the process (especially when working on the same file which really isn’t easy), it allowed remote working and helped us to track the progress the group was making. Using Github instead of Dropbox was a lot better as this ensured that no one could affect the code itself and then it being lost forever as Github allows you to go back to a previous commit that basically backed up each and every implementation made.

6.3. Reflection of Final Version

Firstly, the user has the opportunity to login to the system by accessing the XML files. This creates a stable way to accessing files in a structured way that can be viewed outside of the system for exporting purposes and diagnostic. Another advantage of the way log in is used is the fact it searches through a very easy to read file, which makes the efficiency of the program very fast and uses very little memory reading through the XML file. It iterates through as if it is a big array which allows it to run in the background and memory to be targeted where it needs to be.

A user can also be registered which makes the program into a slightly more user friendly design, as it allows direct input from the user into the system. This adds to the XML file directly which the login function reads from which shows the fact that the system is well integrated with each other. It stores static information and an admin can go through and change the permissions of a user which will open up more options in the system’s menu and allow higher privileges.

A user can book a flight that is valid at any time. This is done by reading a separate XML file with seat allocations into the system and allowing the customer to pick a seat depending on that. This then adds the user to the flight information XML document, which also adds to how integrated the system is. Bookings can be altered by the admins. This was a good example of implementation as it takes different types of information and displays it into the same file without causing issues and altering efficiency.

By having a time board in our software, it allows the implementation of our Quicksort and Queue algorithms that import times of flights, sorts them appropriately and then places them onto a Queue which would, in the real world, be outputted to a separate TV screen as you see in many airports. This is good as it uses a very complex sorting algorithm efficiency and then the queue would be as you’d see on a real life situation so shows the scope of our program. This could be better in some ways due to the fact Quicksort is one of the slowest and hardest to implement, so in moving the system around it could cause issues that we hadn’t foreseen.

The plane data that the software receives is also kept inside of an XML file that incorporates the different rows and columns of the plane and also has storage capacity information and model types. This is imported when an admin creates a flight using that plane data and it is then loaded in and creates a text file that holds seat information.

6.4. Project Evaluation

**What went well:**

In this project we did a lot of things positively and ensured that we worked as a team throughout the process. One of the positives of this task was splitting up the coursework documentation early and looking at our strengths and weaknesses from previous modules and that shaped who did what. For example, Joe was good at Gantt charts and have done them for every single module that required them in the past which led to an easy decision in who was being allocated that task. Also, there was a lack of selfishness in how tasks were allocated and everyone did their fair share throughout the tasks, which helped team dynamics and morale to get things done where required.

Another positive was the use of Github for the code development as well as the organisation of coursework files. This acted as a centralised location for all the work to be submitted with the safety of using the backups if things went wrong, as well as having a safe place to keep the files in case of memory loss. By using this we were also able to have issues generated, which could be assigned to different people and then labelled appropriately which helped organisation and led everyone to understand what they had to do.

The code itself mimics a lifelike problem that has been solved, which helps the development as rational scenarios could be thought up and then this changed the code accordingly. This also allowed us to research how airport systems work and what elements have been used before and which ones should be included in our project.

**What went wrong:**

One of the drawbacks of the program is the fact that it doesn’t have a graphical interface which takes away from the user experience slightly and makes it a little outdated in it’s aesthetic. Despite this, it still has the required functionality for what we set out for, but this would be added later if we had more time. This is discussed below.

Another issue was meeting deadlines that we had pre-defined to try and set up a structure to follow. This became hard when taking into account different projects and coursework hand-ins, which resulted in a lack of communication at certain points of the module and a sudden kick into action about a week before the final hand in. Although the work was eventually done, more time should’ve been given to allow issues to be sorted in the final week rather than a rush of work that could’ve been a lot better. Despite this, the group worked efficiently when it was required and there was an equal amount of work done by all members and it resulted in a very positive working environment to develop a product in.

**More Time Ideas:**

* A GUI that allowed the user to log in and register and have a booking list on the front page. It would also hold information such as flights and online check-in and then would allow the user to book the seat and see a diagram of the plane, rather than 0 and X. This would improve the user experience which is integral for a today’s-world product.
* Instead of using a static data file like XML, something like a more significant database could be used, for example SQL, which would allow easier changes within tables and data and would also be easier to write to, as SQL parsing would be a difficult implementation but would then only have to be called remotely from that point onwards.

7. Appendices

7.A. Use Cases

[TO ADD]

7.B. Flowchart

C:\Users\n0623403\Desktop\Software-Design-Imp\doc\7. Appendix\Flowchart UPDATE.png

7.C. Test Images

[TO ADD]

7.D. Menu Areas

[TO ADD]

7.E. Gantt Chart

Please find in file as BluebrookAiport\_SDI\_SDLC.pdf