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.

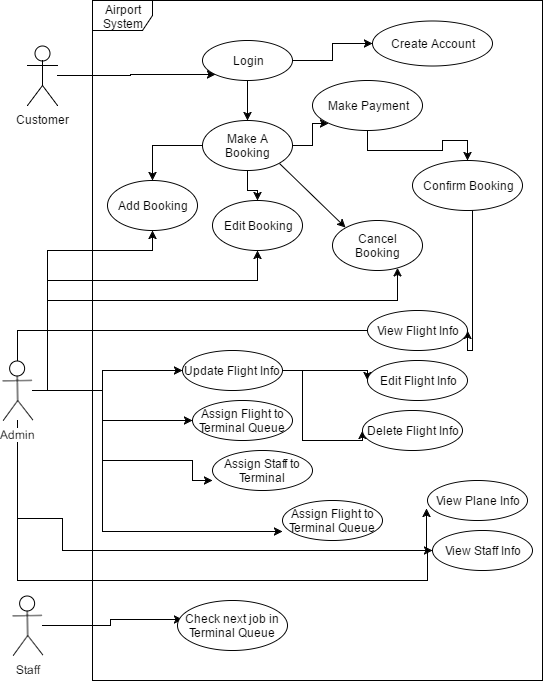
1.1. Functional Requirements

[TO COMPLETE]

2. Requirements In-depth

2.1. UML Use Case Diagram

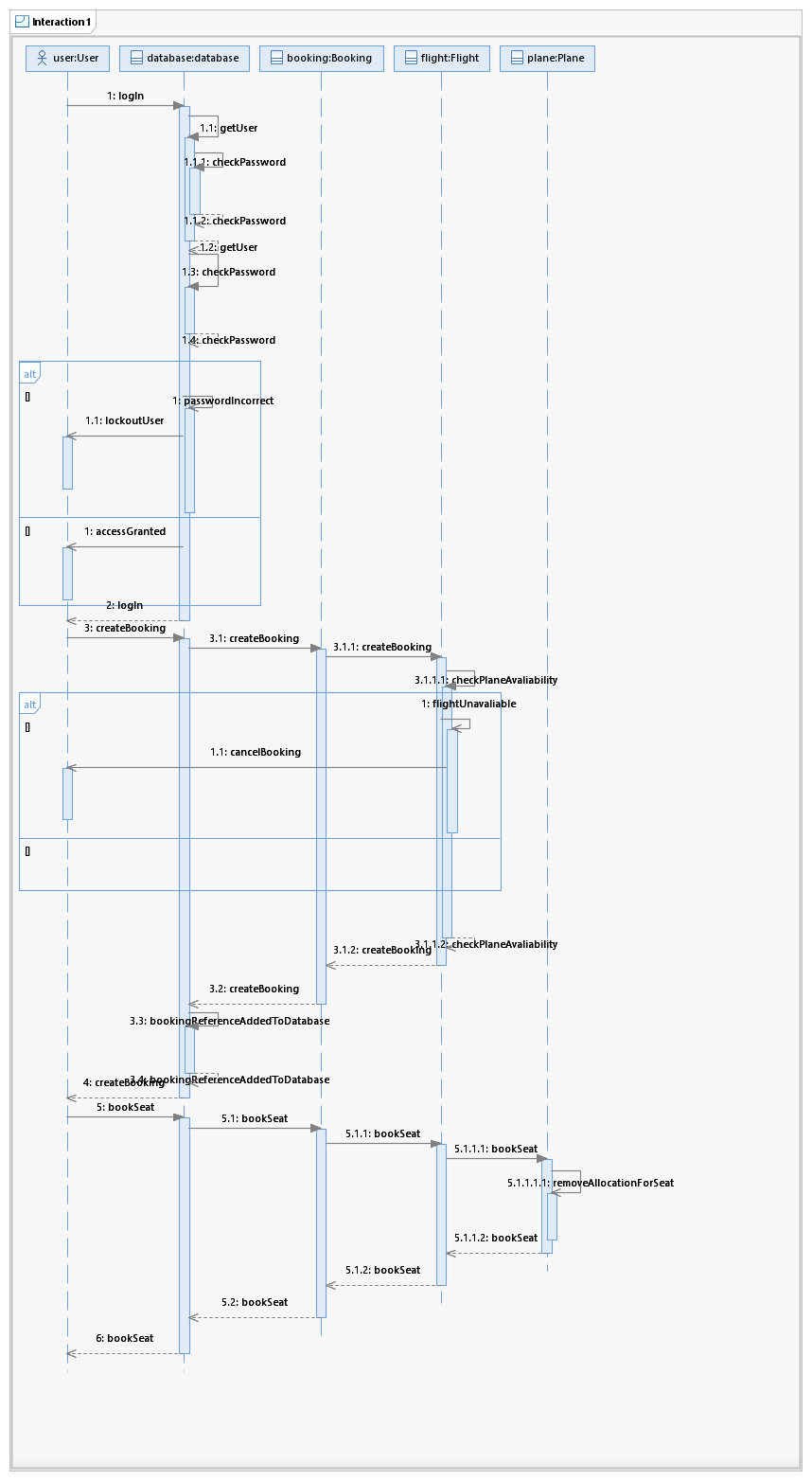
By designing a use case diagram the system is able to be visualised and to see how interactions will be taking place between different elements. For example, you can see how the Administrator has access to very high level functions that the Customer will also be able to see once completing steps beforehand.



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

[TO COMPLETE]

3. Approach

3.1. Chosen Methodology

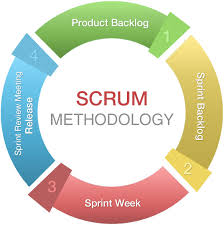
Methodologies are methods which are used to carry out projects and manage development teams in the IT industry. Many different mythologies fall under the agile mythology category. The main difference between agile methodologies and more traditional mythologies (Waterfall for example). Unlike waterfall 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

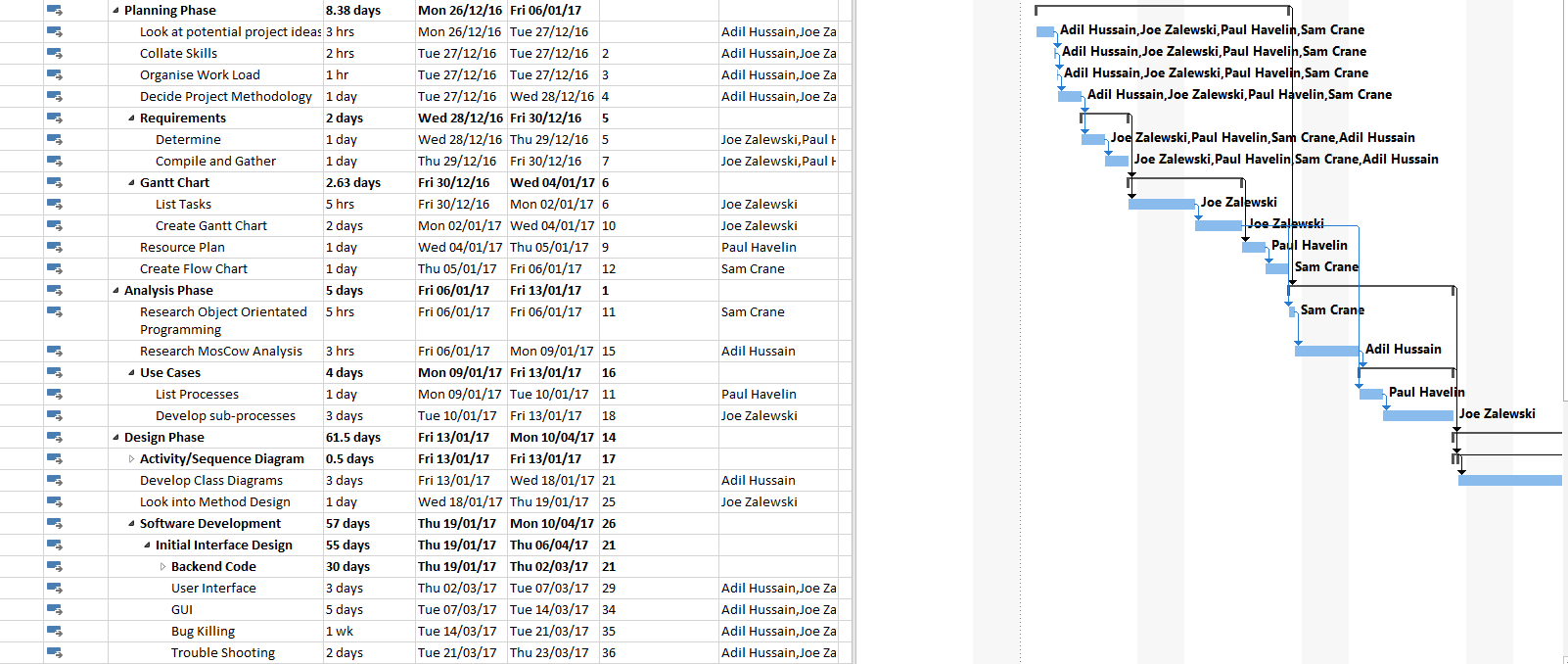
The Bluebrook Airport system utilises an object-oriented approach in its code design. The entire concept of the program revolves around objects interacting with one another, such as users, flights and planes, making object-orientated programming ideal and efficient. Additionally, in a system like an 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

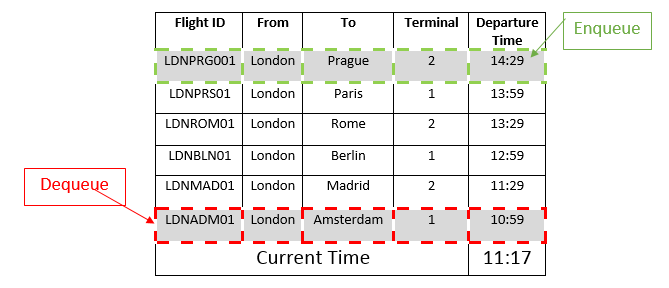
The software development life cycle was created using Microsoft Project in the form of a Gantt chart. The team decided that this was the best option as the software not only allows the visualisation of how the project should be coming along at all stages but also has resource allocation and management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

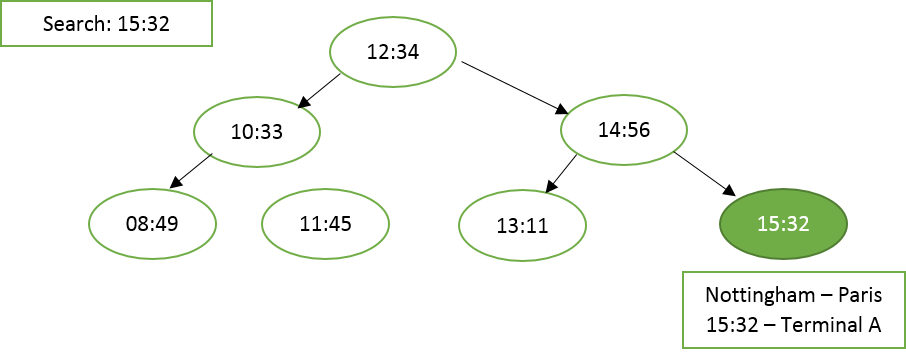
User enters a flight time

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

[TO ADD]

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 Method and Final Product

[TO ADD]

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