*Coursework B Group 2 Report - Check In*

*Simon Grange, Safa Al Ameri, Mohamed Serry, Bruce Tauro, Rania Ballout*

# **12th June 2020**

# 1.Introduction

This involved the addition of threads and design patterns, and the development of a suitable GUI to show the state of the simulation.

## Epic 1 - Core Functional Requirements

These are the core functional requirements met in this assignment. These are addressed first with the extended requirements section below. Table 1 Lists the developments, the primary developer and the state of readiness.The application simulates the arrival of passengers at the airport. This simulation consists of the following;

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Functional Requirements** | **Software Requirements** | **Lead Author** |
|  | Read in details of passengers and flights when it starts up |  |  |
| 1) | Multiple check-in desks |  |  |
| 2) | Single queue of passengers | Random selection of passengers who have not yet checked in |  |
| 3) | Group of flights waiting to depart |  |  |
| 4) | When passengers arrive at the airport, they join the back of the queue |  |  |
| 5) | When they reach the front of the queue, they will be processed by the next available check-in desk |  |  |
| 6) | ‘Check In’ | Each passenger has up to one piece of baggage, and the dimensions and weight of the baggage can be chosen randomly when the customer joins the queue  GUI shows the list of **passengers** waiting in the **queue** and the details of what each **check-in desk** is currently doing |  |
| 7) | Passenger assigned to the appropriate flight |  |  |
| 8) | Check In desk Closes | period of time has elapsed - check-in desks close |  |
| 9) | Flights Depart | planes depart - remaining in the queue will not be able to board their flights |  |
| 10) | Log Generated |  |  |

#### Table 1: Functional Requirements Delivery

# Software Engineering Requirements

## Software Components

The functionality that our system provides meets the specification and is detailed in table 1 above. Extension to the core requirements is detailed separately at the end of this section and the comments in table 1 relating to these are *italicised.*

* Functionality that our system provides;
  + Passenger entry to airport
  + Joins the end of the queue
  + Assign passenger to a check in desk
  + Check in and check Bag
  + Weight
    - ( ) Kg
  + Size
    - Length
    - Width
    - Height

### GUI

This shows the list of passengers waiting in the queue. It provides details of what each check-in desk is currently doing;

1. Processing a particular passenger
2. Charging for excess baggage
3. Status of each flight
   1. Number of passengers checked-in
   2. Weight of baggage *etc.*

The GUI was completely redesigned from stage 1, with completely different code.

### Log

Logging refers to the recording of the application activity. Whilst logging can be used to store exceptions, the primary purpose here is a record of the information, and is extended to provide warnings as messages that occur during the execution of the check-in process.

This records the following events in near real time;

a) Passengers joining the queue

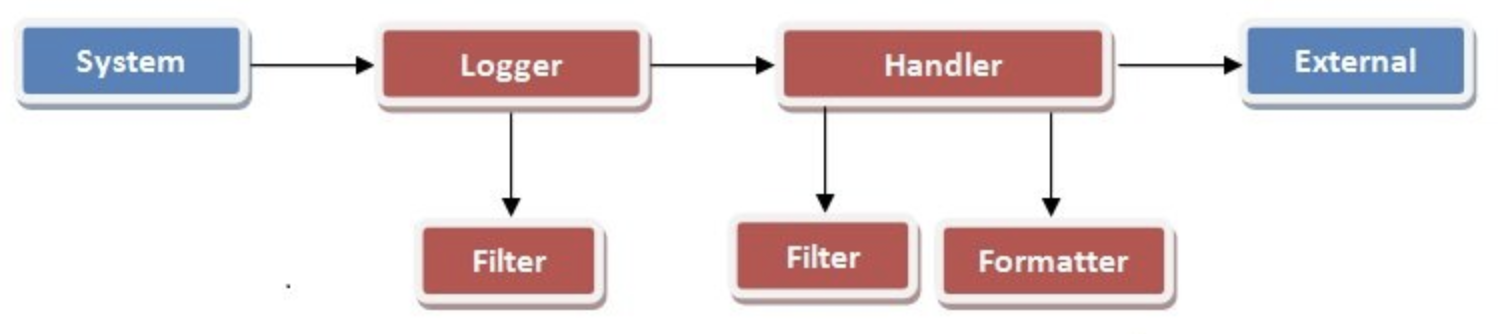
b) Passengers checking in

c) Passengers boarding flights

The output log is written to a file on closing the application. These processes record our application’s activity. Logging is used to store exceptions, information, and warnings. These messages occur during the execution of a program and are reported to a file on closing the check-in process. Whilst logging is used in the debugging process of a program, on this occasion the purpose is to provide evidence of completed activity. Using the java.util.logging package, the [Logger](https://docs.oracle.com/javase/7/docs/api/java/util/logging/Logger.html) object is used to log such messages.

The Logger object generates a [LogRecord](https://docs.oracle.com/javase/7/docs/api/java/util/logging/LogRecord.html) object which stores the message to be logged, which is then forwarded to the [XXXX] handler assigned to the Logger object. The check-in system uses a [Filter](https://docs.oracle.com/javase/7/docs/api/java/util/logging/Filter.html) to structure the log messages. The handler then publishes the logged messages to the output file; [filename].

The Logger allows the application to log independently of where the output is sent. The application logs the message by passing an object (+/- exception with an optional severity level) to the logger object under the identifier; [log id].



**Log Formatter**

The Formatter is an object that consists of taking the binary object and converting it to a string representation for the three records above.

**Appender - Extended functionality**

The appender listens for messages at or above the specified minimum severity level. This includes passengers who missed flights or did not arrive. The Appender takes these messages and posts them appropriately. Message dispositions include:

1. Passengers failing to join the queue
2. Passengers failing to check in
3. Passengers failing to board flights

The output of these [‘warning’] logs

1. Display on the console via the syslog
2. Write to a file on exit

Other options such as appending to a database table, distributing via Java Messaging Services (to the other services), sending via email to notify those who will act on the results or writing to a socket if they have JDBC access to a database recording passenger flow were beyond the scope of this assignment.

### Extending the core requirements

* Constraints
* Needs GUI refinement (currency display)
* Error messages (e.g. >3 letters)

The team experimented with agile methods. The first Epic focused on core functionality and had iterations every two weeks and features were developed in each iteration according to the backlog. Team planning via Webex (Cisco systems, USA) managed the SCRUM meetings where design, development and testing of each iteration was completed before discussing the future iterations. Some refactoring of code was at the end of each iteration based on the most recent ‘pushes’ to branches in the repository..

Our team implements pair programming (SG and AS for GUI development and SG and SA for Log / IO development), ‘stand-up meetings’ were managed virtually via the ‘What’s App’ (Facebook, CA) communications tool for the group - see figure 2 below;

# Design

## Use of Threads

These ensure that they are synchronized where necessary and do not interfere with one another. For the core functional requirements, you should have one thread for each of the check-in desks, and one thread to add passengers to the queue. You are encouraged to add more threads to the application when you develop extensions and also consider more advanced scenarios like the producer-consumer model. Thread-safe versions of some collection classes were not adopted so as to demonstrate knowledge of threading.

## Use of Patterns

The following design patterns were used in our application architecture; Singleton, Observer and MVC.

### Singleton Pattern

Our team employed the Singleton pattern to implement our log class. It is used to ensure only one instance of a class exists in our system, whilst allowing other classes such as the handler and formatter to get access to this instance. The log class is accessed by various objects in a system such as the queue, check in and register of who is eligible to board flights. This is a common use for the singleton pattern with the added benefits of access control for the user to see the reports.

#### Singleton implementation

public class NextNumber {

private static NextNumber instance = new NextNumber();

private int number; // stores current number

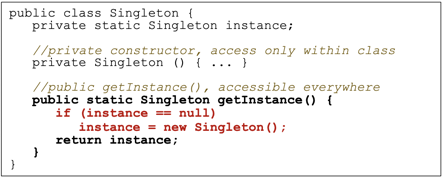
private NextNumber() { this.number = 0; }

public int getNext() { // returns next number number++;

return number; } public static NextNumber getInstance() {

return instance; } }

This also offers lazy instantiation which since the logging system must always generate an output on exit ensures automation of the process.

based on this

### Observer Pattern

The observer and MVC patterns are used in the GUI. The purpose of the observer pattern in this scenario is to define a one-to-many dependency between the objects when multithreading so that when one object changes state, all of its dependents are notified and updated automatically. The main motivation for this is to reduce coupling between the classes and increases reusability as each of these dependents publish their output to the GUI.

Effectively this allows observer objects to register interest in other objects’ state (subjects), so that whenever the subject changes state, the subject informs all the observers.

The [XXXX] interfaces is used to add generality and flexibility, and further reduce coupling between classes, so the subject interface, which is implemented by the [XXXX] subject class updates the Observer interface, which is implemented by each [XXXX] observer class.

Clock class, holding data about time from the queue

3 Observers – objects of the AnalogDisplay, DigitalDisplay, and Counter classes, react when they are told the time has changed so the display objects react by altering their display;

**[screenshot - GUI Queue registration time]**

The counter class increments a counter and shows a message

**[screenshot - Next passenger - check-in time]**

The Subject (here the Clock object) knows who to tell when it has changed? By;

**[source code - updates]**

The Subject class keeps a list of all the objects (Observers) which need to be told that the information has changed

Whilst the Subject class has methods to add or remove Observers from the prescribed list, it remains static in this system.

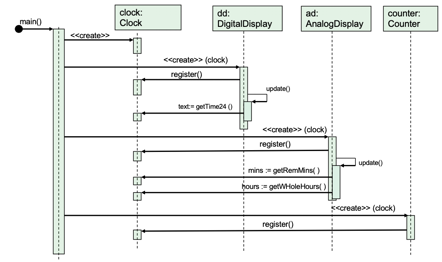


Figure X: Sequence diagram - The observer pattern is used for the time management

### MVC Pattern

The architectural design pattern refers to the compound ‘model, view, controller’ pattern separates a program's data, its display, and how it handles user interaction;

##### The Model

The underlying data of the program and any associated logic (i.e. methods)

##### The View

The visual display/presentation of the model

##### The Controller

Handles user interaction with the model

This pattern says that you should implement the model, view and controller as separate modules that are largely independent of one another. MVC uses three other design patterns, the ‘Observer’ outlined above as a glue between View and Model. The Strategy pattern acts as a glue between View and Controller. Finally the Composite pattern handles interface updates, since the GUI is a composite of components. MVC therefore makes it easy to update the interface of our program, since the interface code is already separated

#### Other Design Patterns

No other design patterns were used in the preparation of this system.

## 

## 

## Packages

The following package structure was adopted;

* Src
  + GUI
  + Logger

## Version Control

Version control was managed via GitHub;

Repository structure -branch management

TypeNameLatest commit messageCommit time

[.idea](https://github.com/raniazqt/HW_Airport/tree/DEV/.idea)[git ignore updates](https://github.com/raniazqt/HW_Airport/commit/549b0b4b57b356b61386689d35a608159efbc924)

[.settings](https://github.com/raniazqt/HW_Airport/tree/DEV/.settings)[ignore eclipse settings file](https://github.com/raniazqt/HW_Airport/commit/75c0d6a078a1e06cb8d6fe376068acdd6acced98)

3 months ago

[resources](https://github.com/raniazqt/HW_Airport/tree/DEV/resources)[Count bug fixes](https://github.com/raniazqt/HW_Airport/commit/397574f949d20692e9b16baeafef94baf131f376)

2 months ago

[src](https://github.com/raniazqt/HW_Airport/tree/DEV/src)[Merge branch 'feature/CheckIn\_Desk' into Queue\_Imp](https://github.com/raniazqt/HW_Airport/commit/5cc1b6625b376e218d2a1cec80583be8f664fa81)

2 months ago

[test/com/hw/airport/service](https://github.com/raniazqt/HW_Airport/tree/DEV/test/com/hw/airport/service)[imports](https://github.com/raniazqt/HW_Airport/commit/0bbc1e2ace017e85de3ab8232872781ea200c7ed)

3 months ago

[.DS\_Store](https://github.com/raniazqt/HW_Airport/blob/DEV/.DS_Store)[Temp GUI placed in Sgui](https://github.com/raniazqt/HW_Airport/commit/4563638e947a474688eeb66b46953965fc8ee4d4)

4 months ago

[.classpath](https://github.com/raniazqt/HW_Airport/blob/DEV/.classpath)[resolve conflict](https://github.com/raniazqt/HW_Airport/commit/d5173e2f619360f9eddc215ad71bacc3b4a65cb6)

3 months ago

[.gitattributes](https://github.com/raniazqt/HW_Airport/blob/DEV/.gitattributes)[Initial commit](https://github.com/raniazqt/HW_Airport/commit/cb090d95c0425509463f4702794cac3418420a3d)

4 months ago

[.gitignore](https://github.com/raniazqt/HW_Airport/blob/DEV/.gitignore)[Merge branch 'master' into Code\_Refactor](https://github.com/raniazqt/HW_Airport/commit/d203556303f91dcadc0a77ea5b88cc591f2a0487)

3 months ago

[img.png](https://github.com/raniazqt/HW_Airport/blob/DEV/img.png)[Code refactor](https://github.com/raniazqt/HW_Airport/commit/9c8a8d7ac90f90737c7431050960e83148f84a90)

3 months ago

[manifest.mf](https://github.com/raniazqt/HW_Airport/blob/DEV/manifest.mf)[GUI adjustment](https://github.com/raniazqt/HW_Airport/commit/fb4a476c35bad015b6c2c9c364c5393f97fcba7b)

3 months ago

Inside ht src branch the core package file structure is as follows;

[GUI](https://github.com/raniazqt/HW_Airport/tree/DEV/src/com/hw/airport/GUI)[Queue](https://github.com/raniazqt/HW_Airport/commit/3e92181664bc43698f0594f4f6ead8a90d2f5546)

2 months ago

[exception](https://github.com/raniazqt/HW_Airport/tree/DEV/src/com/hw/airport/exception)[added unit tests, exceptions, code comments.](https://github.com/raniazqt/HW_Airport/commit/0723e7adba69b2aa63abc8e58b0a7344f8e240fe)

3 months ago

[main](https://github.com/raniazqt/HW_Airport/tree/DEV/src/com/hw/airport/main)[Queue](https://github.com/raniazqt/HW_Airport/commit/3e92181664bc43698f0594f4f6ead8a90d2f5546)

2 months ago

[model](https://github.com/raniazqt/HW_Airport/tree/DEV/src/com/hw/airport/model)[Merge branch 'feature/CheckIn\_Desk' into Queue\_Imp](https://github.com/raniazqt/HW_Airport/commit/5cc1b6625b376e218d2a1cec80583be8f664fa81)

2 months ago

[service](https://github.com/raniazqt/HW_Airport/tree/DEV/src/com/hw/airport/service)[Merge branch 'feature/CheckIn\_Desk' into Queue\_Imp](https://github.com/raniazqt/HW_Airport/commit/5cc1b6625b376e218d2a1cec80583be8f664fa81)

2 months ago

[util](https://github.com/raniazqt/HW_Airport/tree/DEV/src/com/hw/airport/util)[add the code to load flight and booking data from files](https://github.com/raniazqt/HW_Airport/commit/0565882a626693cb4345e4aaab136bdf10043ecf)

3 months ago[.DS\_Store]

(<https://github.com/raniazqt/HW_Airport/blob/DEV/src/com/hw/airport/.DS_Store)>[Temp GUI placed in Sgui](https://github.com/raniazqt/HW_Airport/commit/4563638e947a474688eeb66b46953965fc8ee4d4)

### Delivery

The final application was exported to a jar file, to run. Due to the change in program requirements consequent on the COVID-19 Pandemic restrictions our group will demonstrate virtually by submission of the necessary materials.

Jar file at;

**[Address]**

## Epic 2 - Extended Functional Requirements

You are also asked to extend the core requirements.

Marks will be awarded based on the complexity of your extension(s)

knowledge and understanding required to implement them.

1. Allow the user to alter the speed of simulation using runtime controls.

2. Allow the user to open or close individual desks as the simulation proceeds, or the simulation could automatically open new desks (up to a limit) when the queue gets to a certain length.

3. Your simulation could have more than one queue, with each queue feeding passengers to more than one check-in desk. You could limit the size of queues. You might introduce different cabin classes (e.g. economy, business) and have queues with different priorities for these.

4. Rather than the check-in desks closing at a particular time, each flight could have a specified departure time, and check-in desks will only accept passengers who arrive before their flight departs. You could show flights departing by removing them from the GUI.

5. If you’re feeling even more ambitious, you could simulate more details of the airport experience, e.g. queuing for security. However, bear in mind that a small number of good quality extensions are better than lots of rushed extensions.

**Group Report**

A stage 1 / 2 comparison reflects the necessary changes in work practice from a ‘waterfall’ traditional method to an agile one. These are outlined in the table 2 below;

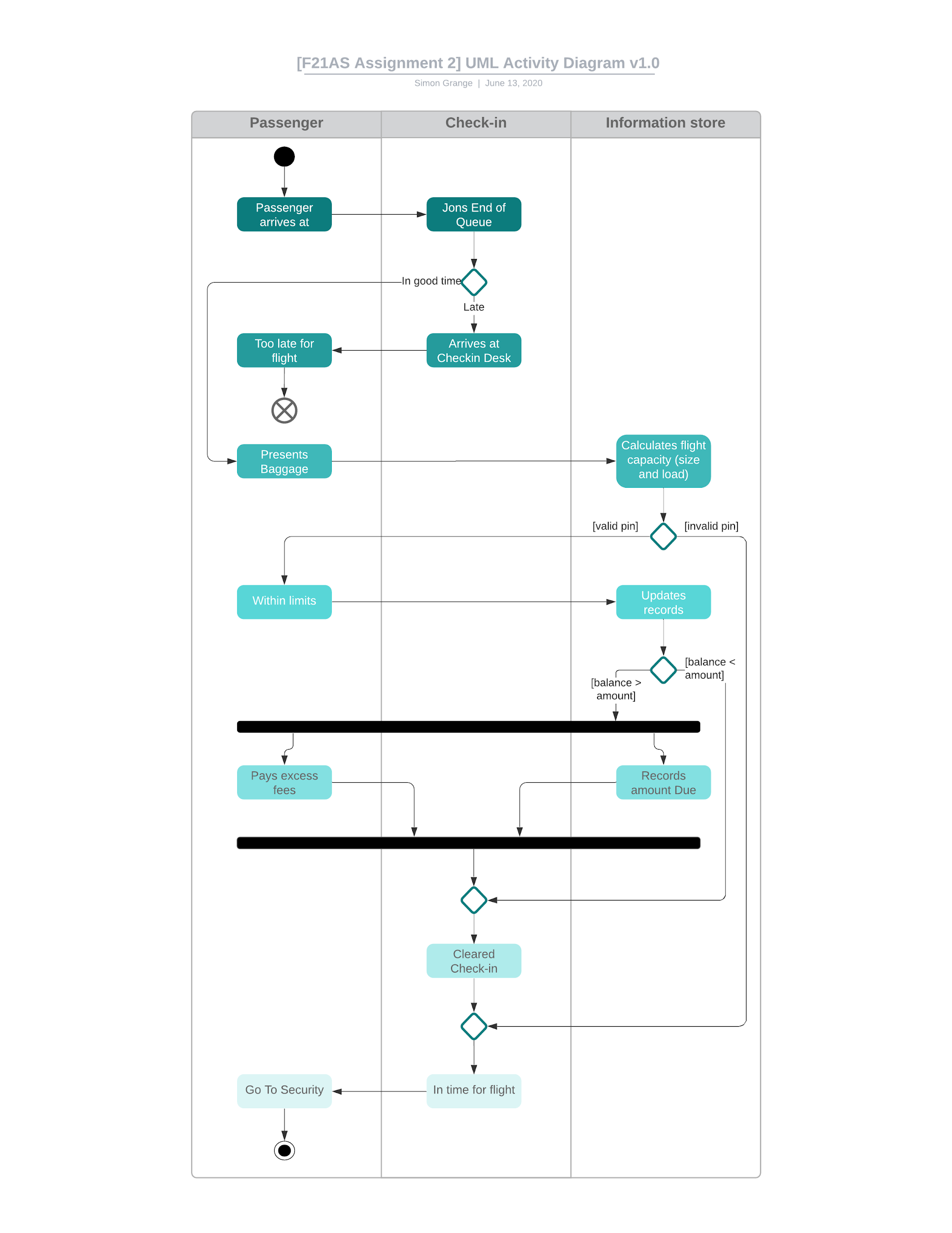
The aim is to move away from the mythical man month (Fred Brooks)

|  |  |  |
| --- | --- | --- |
|  | **Stage 1 approach** | **Stage 2 approach** |
| **Advantages** | Face-to-Face team building, made workshops with whote boards easier and more spontaneous | Completely virtual infrastructure allowed the project to continue under the terms of strict physical isolation of staff members |
| **Disadvantages** | Rigid approach generated ‘time crunch’ when deadlines were superimposed from independent activities such as other projects. | Time frames less precise, agile methodology maintained activities through multithreaded linear progression of service development |

#### Table 2: Comparison of Stage 1 and stage 2 work methodologies

Sections of agile technique are detailed in [section XXXX] below.

## UML Diagrams

****

#### **Figure 1**: **[F21AS Assignment 2] UML Activity Diagram -** This shows the main activities of the workflow, and the contents of each stage of the process

#### These show the associations between the top levels of classes for the Check-in process with related classes, and the contents of each class being detailed in Appendix I.

#### **Figure 3:**

#### Figure 2: **[F21AS Assignment 2] UML Class Diagram in code -** This shows the associations between the classes, and the contents of each class

**UML (lecture notes)**

# Project Management

## 

## Agility

According to the Manifesto for Agile Software Development:

•We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

•Individuals and interactions over processes and tools Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan

•That is, while there is value in the items on the right, we value the items on the left more.

[see www.agilemanifesto.org]

In the spirit of this, and using ‘state of the art’ methods; our team developed the program using agile processes. JIRA was adopted as the established shared sprint work environment. Details of this are available at; [JIRA view only address with name and login?]

### Agile techniques and tools

Flexible hierarchy;

* RoadMap
* Epics
* Storyboards
* Stand Up Meetings
* SCRUM

#### RoadMap

[Screenshot - JIRA]

#### Epics

#### Storyboards

#### 

Our group did not rely on a roadmap for a short project like this as the specifications were clearly defined in the scope.

**Epics**

There were two main phases of the coursework and of the second part there were two epics;

Epic 1 - Related to the core functionality

Sprints

Sprint 1 - Design of the application including

Sprint 2 -

Sprint 3 -

Epic 2 - The additional extensions to the core program.

Sprints

Sprint 1 -

Sprint 2 -

Sprint 3 -

Version Control was managed via GitHub

Agile techniques explored included pair programming (Simon Grange & Mohamed Serry - GUI, and Safa Al Ameri & Simon Grange Log I/O) stand-up meetings (performed via Webex) and time-boxing using separate Sprints. This was necessary due to the team being under lockdown for the duration of the project in separate cities due to the presence of the COVID-19 pandemic. This imposed constraint demonstrated the effectiveness of the methods employed to deliver management services for the software development.

**Observations**

4. An explanation of how and where threads are used in your application.

Threads

Multithreading for different check-in desks

6. Sample screen shots.

Screen shots of different stages of the application

7. A brief comparison of your development experiences in Stage 1 and Stage 2.

Did plan-driven or agile development work better for your group?

plan-driven (Pre-COVID) worked well with regular meetings 'in the office' but would not have worked during the COVID-19 outbreak as it was necessary to have irregular meetings due to different commitments.

What problems did you encounter, and what might you do differently next time?

In future the model of using

GUI design can use dedicated GUI generation tools

Diagrams

UML diagrams where appropriate

snippets of code where relevant

Patterns

Singleton

Observer

#### Stand Up meetings

#### 

#### Figure 2: ‘Stand-up’ meetings managed via ‘What’s App’.

With regular time-boxing, though, the latter became somewhat practical to use for our project due to the changes in project schedule and remote working restrictions.

#### SCRUM

Sections to Integrate in the final report

# Functionality

## Core requirements complete

## Extensions

## User Interface

## Demonstration

### All features shown to work in demo

# Implementation

## Code quality,

### Readability

### Comments

# Software Documentation

## Technical Writing

## UML

# Report

## Iterations

## Development Tools

### Agile techniques

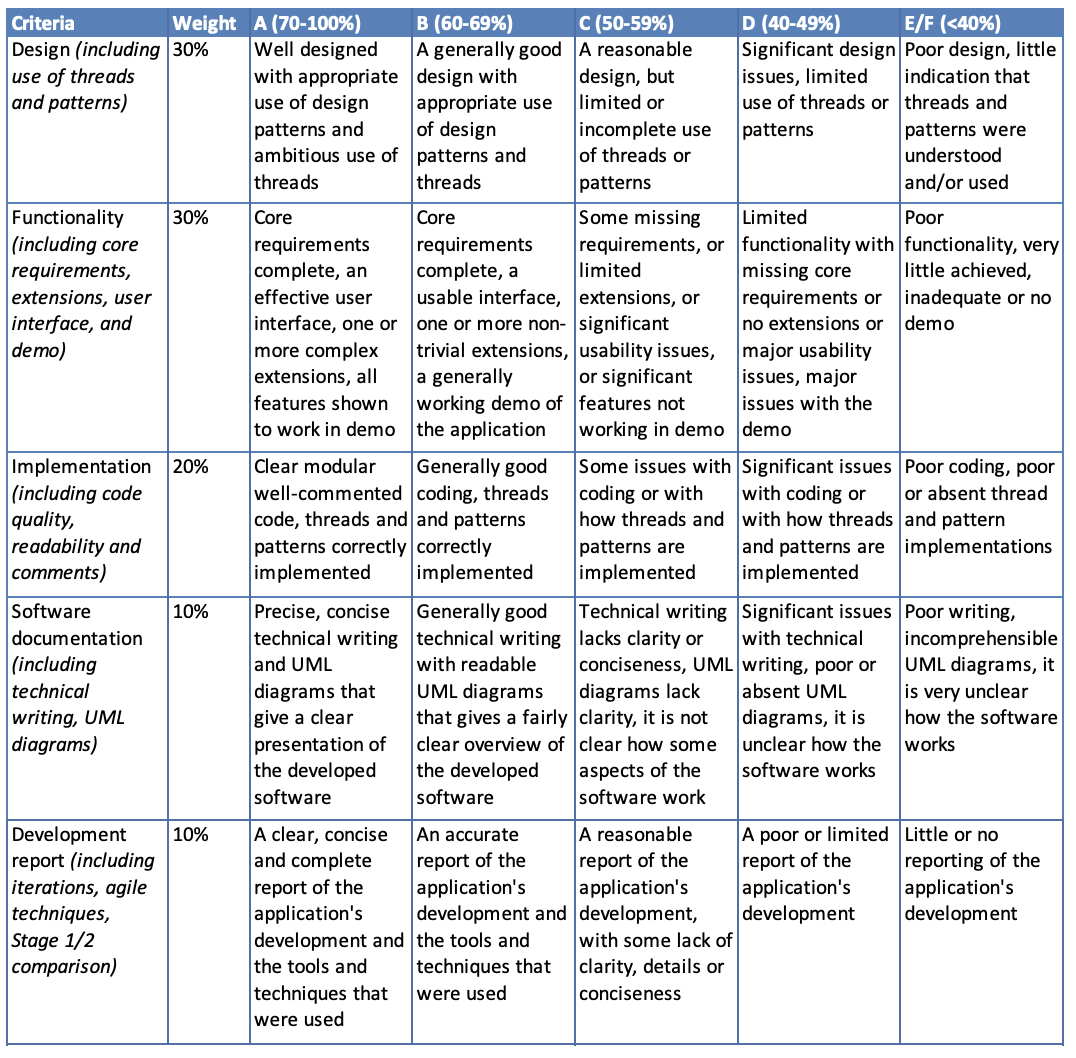
## Stage 1 to 2 comparison

# Appendix I

## UML Diagrams

The UML class diagram for the singleton pattern employed in the logging system allows for a single point to access it, and in so doing controls access to the instance. This avoids the use of a global variable.

# Appendix II - Marking criteria



**Unlinked References**