COMP2211 Final Report - Group 2

Eren Rafet (er10g23), Bozhang Wu (bw9n21), Louis Townsend (lmt1n22)
Abdullah Hariry (ah4u22), Hossameldin Tammam (htft1e22)

June 15, 2025

1 Evaluation of teamwork

1.1 What went well

There were many strengths in our teamwork throughout the project. Our main strength was our communication and attendance to meetings. Generally, everyone was in attendance or eager to catch up if they'd been absent. As a result, we were able to discuss and effectively plan and conceptualise the project collaboratively throughout the entire process, ensuring that the project was the result of everyone's contribution.

Team members demonstrated good responsibility for their assigned tasks, with Louis, Eren and Hossam in particular working the most consistently throughout the project lifecycle. This reliability created a stable foundation for our development process and helped maintain momentum even during challenging phases of the project.

Furthermore, when delivering on tasks, all group members were receptive to feedback and constructive criticism on their work. This openness to peer review allowed us to identify potential issues early and improve our deliverables through iteration. Implementing this collaborative approach enabled us to ensure that everyone's features integrated as seamlessly as possible, resulting in a more cohesive application.

1.2 What could have been better

Task definition and clarity were our primary areas for improvement. Due to the group's lack of experience with the software development process, we failed to provide sufficiently detailed task specifications during the planning process. This lack of clarity led to several issues, including team members starting work without fully understanding the requirements and multiple instances of work having to be redone by other members when the features didn't meet the required specifications.

Furthermore, some group members were unfamiliar with git version control and did not attempt to learn it independently. This created an uneven distribution of work, as conflicts on git merge requests into the main branch had to be resolved by those familiar with the tools. This technical burden fell disproportionately on Louis and Hossam, who dedicated a significant amount of time to resolving git-related issues that should have been distributed across the team.

More proactive communication from those group members who were unclear on their tasks or how to use the chosen development tools would have made the development process significantly more streamlined. Instead, a large portion of the first two development increments was devoted to uncovering and then fixing those issues, delaying meaningful progress on the project's core functionality.

1.3 Advantages of agile methodology

The use of the agile methodology in our development process encouraged an incremental approach to our solution. This incremental approach was the key to our success as it enabled us to focus on completing the software in stages, ensuring each feature we delivered was polished before moving on to the next feature. In contrast, without the agile methodology, it is easy to fall into the trap of attempting to complete the project as a whole and not breaking the project down into manageable chunks and thus failing.

Furthermore, holding sprint review meetings with our supervisors was useful as it provided immediate feedback on our work. This enabled us to stay on track for final delivery, and ensured that we spotted errors or areas for improvement early and corrected them before they affected the project too drastically.

1.4 Disadvantages of agile methodology

While agile methodology gave us a solid framework for project management, there were some aspects which we saw as a disadvantage. The primary disadvantage of the methodology was the emphasis attributed to daily scrums. In a dedicated, 9-5 software development role, daily scrums may well be a useful part of the development process. However, at university, where every group member has three other modules in addition to SEG, the use of daily scrums was diminished and we believe a disadvantage. Dragging everyone into a small meeting everyday takes time away from code development, and increases stress as people had to try to find time in the day for the scrum. As a result, we stopped with scrums relatively quickly, resorting to almost daily Whatsapp messages to keep everyone updated.

1.5 Following the XP values

We completed our project according to the five main principles of the Extreme Programming (XP) Methodology. The following is an explanation of how these principles aided the development process.

1.5.1 Communication

Software development is inherently a group activity that relies on good communication. During the project's development, we used various communication software to constantly share information, allowing the team to catch errors early and resolve them immediately. Communication was generally good, although occasionally untimely and unclear, but we did well overall.

1.5.2 Simplicity

We followed the principle of simplicity in our development. We broke the project brief down into sub-problems to enable us to identify the most essential features. From there we identified the main features which were absolutely necessary for a successful project and focused on those to avoid getting sidetracked.

1.5.3 Feedback

The principle of feedback is that by gathering feedback on previous work, the team can identify areas for improvement and then iterate on the product. We improved the application and reports based on feedback from our supervisors as well as internal feedback from group members.

1.5.4 Courage

Extreme Programming defines 'courage' as daring to tell the truth or admitting a mistake without making excuses. There were times when direct constructive criticism was used to highlight when group members had not met the expected pace or not delivered a feature properly. This feedback was always met with understanding and consistently led to improved performance in the next increment.

1.5.5 Respect

Our team members respected each other and communicated fairly. As a group we had an overall friendly atmosphere for the lifetime of the project.

2 Time expenditure

Throughout the development of this application, time was allocated across planning, implementation, debugging, and documentation. Weekly meetings were held to align goals, assign tasks, and discuss progress. The time spent varied depending on the nature of the work. Some tasks were underestimated while some were over estimated.

2.1 Time spent per person

- Abdullah Hariry - 30 hours
- Eren Rafet - 31.5 hours
- Hossameldin Tammam - 34 hours
- Bozhang Wu - 21 hours
- Louis Townsend - 33 hours

2.2 Most costly activities

In general, the activities that were the most expensive in terms of time turned out to be the ones that form the building blocks of the solution, as expected. Tasks involving the modelling and rendering of airport and runway information were consistently the most expensive across the increments, ranging from 8 to 14 hours for each increment, as per our sprint backlogs available in previous documentation. Therefore, it would be fair to say that the visualisation component of the project was the most expensive feature we had, as it required continuous development across sprints rather than being completed in one go.

Nearly all the remaining time spent belonged to features that were completed within single sprints. The most expensive of these was implementing the database, sitting at 15 hours estimated (11 hours actual, implementation took less time than expected). This was followed by information exporting with 14 hours. All the remaining tasks took 10 hours or less.

The cost of these tasks usually came from the fact that we were out of practice with some of the tools that we had to use to tackle them, e.g., JavaFX for visualisations. A lot of time was spent relearning these tools in order to obtain a result that met expectations. Another reason was the amount of time spent on determining how to tackle the activities before actually getting them done, e.g., learning how calculations worked and working on the database design.

2.3 Type of Estimation Used

For our initial estimations, we used **T-Shirt Sizing**, to categorize user stories based on their relative complexity and effort. In the first increment, each user story was assigned a T-Shirt Size (S,M,L,XL), which provided a understanding of its difficulty. Once sized, we converted these T-Shirt sizing into approximate hour ranges based on our shared understanding of team capacity. For example, a task with size small would be translated to 2-4 hours of work while on the other hand, a large task would be 8-12 hours.

Over time, our use of **hour-based estimation** became more accurate as we took into account the experience and skill level of the team member responsible. Each person would provide their own estimate, offering a more realistic perspective on how long the task would take.

2.4 Was It Useful?

The combined use of T-shirt sizing and hour-based estimation was helpful for planning sprints and understanding the scale of upcoming tasks. The value of this approach was most evident in areas like **runway visualisation** and **multi-user login**, where breaking down tasks helped highlight dependencies on other tasks/previous work and allowed for realistic goal setting. In particular, it helped identify tasks like obstacle rendering (estimated at 4h) and user database design (5h) as early priorities, allowing parallel workstreams to proceed efficiently.

However, estimation was less effective for unfamiliar tasks or those with vague requirements. The **"Information exporting"** task, for instance, was estimated at 14 h but required 15 h in practice, mainly due to the additional time needed to debug XML handling and formatting/saving issues. This showed that the accuracy of the estimation depends heavily on how well the task is understood in advance and what classes / techniques were needed.

2.5 Did Estimation Improve?

Yes, our estimation skills improved as the project progressed. In earlier sprints, we consistently underestimated tasks involving GUI elements or unfamiliar tasks we havent worked on before, such as some JavaFX elements, and linking database information. For instance, the task "Add lines to scale for declared distances" was estimated at 4h but required 6h due to unpredictable rendering issues.

Over time, we began determining our estimates more realistically. By the final sprint, the gap between estimated and actual hours reduced significantly. For example:

- The admin dashboard backend was estimated at 10h and completed in 9h.
- Finalising views was estimated at 10–14h and completed in 10h.
- The help documentation was delivered exactly as planned (10h estimated, 10h actual).

2.6 Balancing workload

Workload balancing was not as easy as just allocating specific amounts of work to each individual and hoping it would result in equal contribution. We had to consider different factors, including the ability levels (the most important factor, perhaps), roles for the given increment etc.

We were aware that our team would be formed of individuals of varying abilities, and ensuring that we focused on these members' strengths whilst not overwhelming the more competent members was a difficult strategy to implement and maintain.

Whenever we assigned work amongst ourselves, the Scrum Leader for the given increment made sure to ask everyone in the group if they had any preferences for the upcoming sprint to make the process of assigning tasks easier. Most of the time individuals were willing to complete the task that they had been assigned, but giving group members the option to work on something based on their preference enabled us to achieve higher quality output.

2.6.1 Unbalanced workload allocation, how we dealt with it

We aimed to be as ambitious as possible whilst trying to keep our work distribution across the sprints even for each group member. This meant that each member should have been able to complete their tasks within the given time frame. However there were some unforeseen circumstances that resulted in some members taking over the work that was initially assigned to others.

Sometimes this problem was easier to solve, for example we started leaving some leeway in our sprint planning, which meant that we could push these tasks back to the following sprints. However, this was not always the case. Some tasks had to be done before the other members could carry on with their work, or they had to use that task to confirm that a feature that we were working on was implemented as intended. These inconveniences often slowed our overall progress and depleted the morale of members who were taking on the burden.

An example of this problem occurred in the final sprint. Despite the requirements for the help document and user guide being made clear multiple times, there was confusion on what was required for the task. Due to this and bad communication, the member who was assigned to the task did not complete the work, resulting in Louis having to rush to create the user guide with two hours left to final submission.

3 Tools and communication

3.1 Software Tools

The following table lists all the software tools used in this project and provides a definitional description and value statement for each tool. In addition, the overall effectiveness of each tool has been assessed based on usefulness and frequency of use. (Categorised as high, medium or low)

Name	Description	Value	Effectiveness
Outlook	Outlook is a Microsoft Office application that combines email, calendar, and contacts.	We used Outlook to schedule in-person meetings frequently so group members could view the time and place on the calendar. We also used Outlook to email our supervisor for help or to arrange meetings.	HIGH
Overleaf	Overleaf is a cloud-based LaTeX editing platform that supports multiple people's real-time collaborative writing and editing of documents.	We used Overleaf to prepare reports. Team members could edit online simultaneously, improving productivity and producing a neat and concise document using LaTeX layout.	HIGH
WhatsApp	WhatsApp is an instant messaging software that supports text, voice, and video calls and allows users to share multimedia files between smartphones.	We set up a group chat via WhatsApp for the daily communication of the group members. It was an indispensable tool for group communication.	HIGH
IntelliJ IDEA	IntelliJ IDEA is a powerful Java-focused integrated development environment (IDE), with intelligent code completion, syntax checking, and unit testing.	IntelliJ was our primary tool for code development. It is optimised for the Java programming language and provides many features to streamline development. We used it for writing code, testing, and more. It is an essential tool for software development.	HIGH
Maven	Maven is a project management and build automation tool primarily for Java projects that unifies the management of project dependencies, build process, plugin configuration, and lifecycle.	The project's complexity resulted in the need for many additional dependencies, and we used Maven to simplify the build process, automate the download of dependency libraries, perform tests and package deployments, and facilitate team collaboration and project standardisation.	HIGH
Git & GitLab	Git is a version control system (VCS) for recording changes to sets of files. GitLab is a web-based Git repository manager that allows multiple people to collaborate on a Git project.	We used Git to create local branches, work on them, and upload them to a remote repository when finished. We also managed the project, used version control, reviewed code, and merged branches on Gitlab. Git and GitLab help us with efficient collaborative development and automated operations management.	HIGH
Lucidchart	Lucidchart is a cloud-based visual diagramming and collaboration tool.	We used Lucidchart to draw UML diagrams, case diagrams, and others. It is user-friendly, rich in templates, and a convenient drawing tool.	MEDIUM
JUnit	JUnit is a widely used open-source Java unit testing framework.	We used JUnit to test after each sprint phase task is completed to determine whether the expected and actual results are consistent and to ensure that the program application correctly performs operations.	MEDIUM
Microsoft Teams	Microsoft Teams is a team collaboration platform that integrates chat, video conferencing, file sharing, task assignment and other features.	We used Teams for online meetings when we couldn't attend in-person and to use screen-sharing to get the team to work together on a problem when it arises. We also used Teams to contact our supervisor.	MEDIUM
Jira	Jira is an online project management and issue tracking tool. It also provides ready-to-use Scrum Boards.	We used Jira in Sprint 1 to organise the project tasks, which contained all the tasks to be completed in that phase and were assigned to each team member. However, Jira is a complicated interface with many functions and configuration items for new users. Therefore, we abandoned the use of Jira in the later sprints.	LOW

3.2 Inter-Group Communication

3.2.1 Medium

WhatsApp and Teams were used for communication between group members.

3.2.2 Information Exchange

After assigning tasks in offline meetings, each group member understands what they are supposed to do, and we use the two daily communication software mentioned above to communicate. We can promptly communicate our progress and needs both online and offline. Although there were occasional unclear communications, the team members could make up for it promptly and did not let communication problems affect the project's progress.

3.2.3 Meeting Frequency

There would be fewer meetings at the beginning of each sprint, mostly simple communication on WhatsApp. Towards the end of the sprint, we met every day to ensure the project gets done. The frequency of our meetings is appropriate.

3.3 Group-Supervisor Communication

3.3.1 Medium

Teams and Outlook were used to communicate with out supervisor.

3.3.2 Information Exchange

We arranged weekly meetings with our supervisor and review meetings at the end of each increment via Outlook. In the meetings, we reported to our supervisor, which provided a clear picture of the project's progress. He gave us feedback and advice, but we rarely asked him for help between meetings. Overall, our communication was efficient and pleasant.

3.3.3 Meeting Frequency

We had various meetings with our supervisor at the frequency required by the project, but we did not have in-person meetings with the supervisor during the Easter holiday.

4 Advice to next year's students

Being familiar with git would definitely be helpful, version control becomes essential as more people touch the code. Making sure everyone pulls before working on code and pushes once every day. This helps to costly conflicts
Communication is just as important as coding . If you're unsure of something, speak up early. No one expects perfection, but silent problems usually become team problems later.
Finally, know your strengths and weaknesses. Play to your strengths, but be honest with what you need help with. Supporting each other is what gets the project over to the finishing line.