SEGP 23 Deliverable 5

Williams, Ryan (rw2g21)
Gabaldon Garcia, Victor (vgg1n21)
Booth, Alexander (ab24g21)
Hodgkinson, Roman (rh5g21)
Arora, Rishabh (ra5g21)
Wang, Ziheng (zw9n20)

September 21, 2025

Contents

1 Evaluation of Team Work

1.1 What went well

While evaluating our team's work, we recognised several aspects that went well and areas where improvements could have been made. Firstly, the communication within the team was excellent, fostering a collaborative environment where ideas were shared freely and everyone felt comfortable voicing their opinions. We organised several in-person meetings in the first few weeks of the project to ensure that everyone understood how airport runway systems work. Additionally, our team demonstrated a high level of adaptability, swiftly adjusting to changing requirements and priorities.

Despite the challenges we faced, we delivered a high-quality, working final product which our supervisors described as "sleek" and "professional" in the final marking meeting. We implemented all 23 user stories that we had written, apart from one, which was marked as "Could" and did not impact the overall functionality and goals of the application. In the end, we have produced an application that is easy to use, aesthetically pleasing, and could be used to land a plane safely at an airport, which is something we are proud of as a team.

We have all learned new skills and gained more experience writing JavaFX programs, but most importantly, we gained real experience working on a software engineering project in a team, emulating the professional work environment that we will be joining soon. Before the beginning of this module, no one in the team had ever been involved with a project using the Agile methodology. We have learned fast and incorporated the valuable feedback given by our supervisor. We are now confident that we could perform well in a team using the Agile methodology in the real world, delivering professional work to clients, not just in terms of the final product, but also in any progress reports or other documents that must be delivered.

1.2 What could have been better

One area that could have been improved was time management. Although we could meet most of our deadlines, there were instances where we encountered delays due to unforeseen challenges or inadequate allocation of resources, partly due to unrealistic expectations. For example, during the first increment, we dramatically underestimated how long it would take to get the basic framework (i. e., build script and JavaFX window) to an operational stage, making it infeasible for multiple team members to work on the tasks at once. Therefore, we could not all begin to work on the user stories allocated to that sprint. As such, several tasks were re-prioritised from the first increment to the second when it became apparent that their completion in time before that deadline was impossible. This highlighted the need for improved planning and a more efficient task distribution system for future increments.

Regrettably, a few of our team members experienced unexpected personal circumstances that hindered their capacity to fully contribute. Following a constructive conversation with our supervisor, the consensus was to redistribute tasks amongst the remaining team members. As a consequence, the workload for some increased significantly, as illustrated in Figure 1. Despite this challenge, our collective resilience shone through, culminating in the delivery of a high-quality output. We only fell short in accomplishing a single "Could" user story, a testament to our ability to adapt and overcome in the face of adversity.

1.3 Analysis of Agile methodology

Adopting an Agile methodology for our project had both advantages and disadvantages. One advantage was increased flexibility. Agile allowed us to respond to changes quickly and effectively, ensuring the project aligned with our client's needs. It also facilitated regular feedback loops, enabling us to incorporate stakeholder input throughout the development of our software. On the other hand, the main disadvantage we faced was the potential for scope creep. As Agile encourages iterative development and frequent iterations, there is a risk of constantly expanding the project's scope beyond the initial requirements. We had to maintain a balance between accommodating client requests and ensuring project feasibility within time and resource constraints.

1.4 XP Values

The values of communication, simplicity, feedback, respect, and courage were consistently emphasised and practised within our team. Transparency and open communication allowed us to share knowledge, resolve conflicts, and improve maintainability. Simplicity guided our development approach, keeping the codebase fairly clean, readable, and maintainable. Feedback loops facilitated continuous improvement and encouraged team members to take risks and explore innovative solutions. Lastly, respecting each other's expertise and ideas fostered a positive team dynamic and effective collaboration.

2 Time Expenditure

2.1 Time spent per week

The weekly time commitment of each member was influenced by various factors such as their specific tasks for the sprint, deadlines in other modules, and personal matters. Some weeks, the workload was heavier, requiring more than 20 hours of work, while other weeks were lighter, with team members spending less than 10 hours on the project. During our final hand-in, the required time expenditure increased heavily in order to catch up with the tasks from earlier sprints that were reprioritised, and also to produce the final report, as scenario testing could not be carried out until the code-base was in its final state, in case a regression occurred.

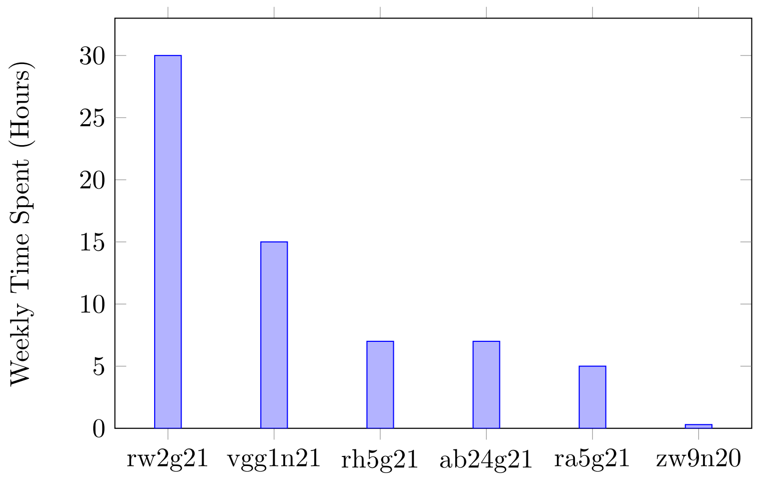

Figure 1: Bar chart of average (mean) approximate hours spent per week on the project

2.2 Burndown charts

A burndown chart is a diagram that represents the work left to do and the estimated time that it will take at each point on the sprint. Please refer to figures 2, 3, and 4 for the burndown chart of each increment.

2.3 Most expensive activities

The most expensive stage in terms of time and effort was the last hand-in, where we had to catch up on many missed tasks from previous sprints as a result of having to reassign many tasks from other team members. A significant amount of time was also spent debugging our code, which proved to be harder as the project got more complex, with an increasing number of "moving parts".

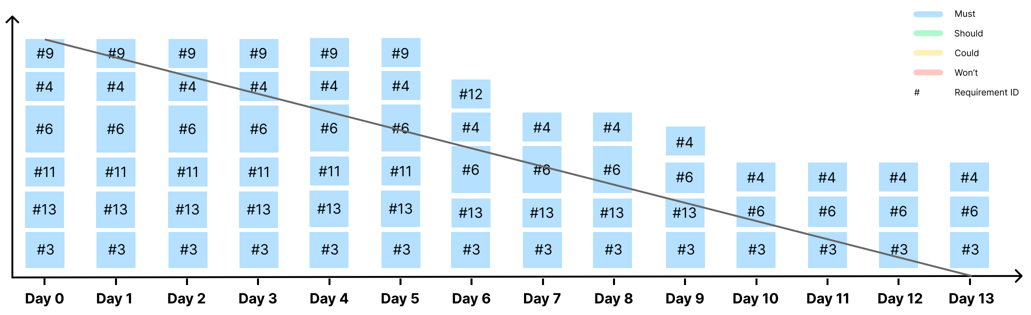

Figure 2: Burndown chart for Increment 1 (created using Figma)

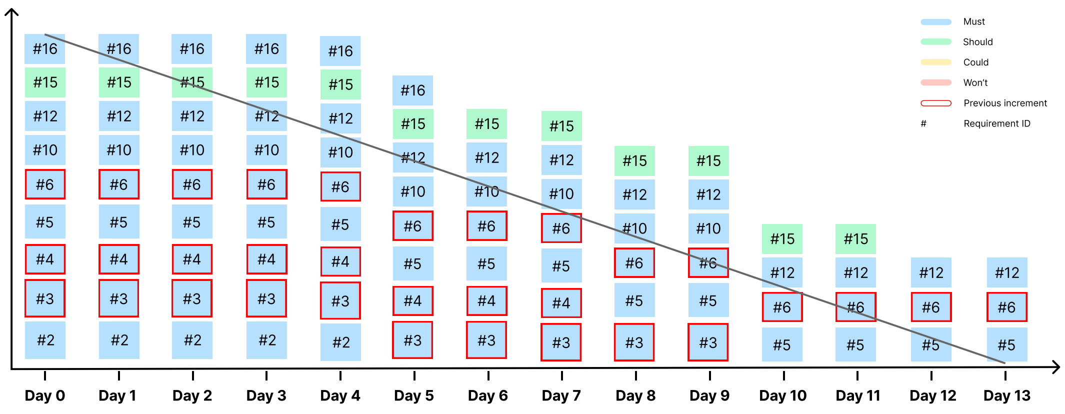

Figure 3: Burndown chart for Increment 2 (created using Figma)

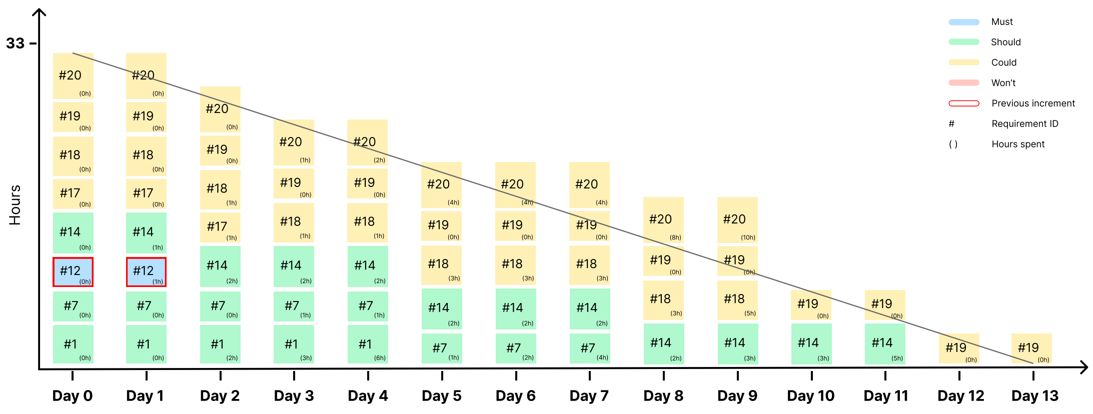

Figure 4: Burndown chart for Increment 3 (created using Figma)

Serialising and deserialising components such as airports, obstacles and application logs to XML files also proved to take a long time, as different (de)serialisation code was required for every component. In addition, the specification required us to use XML files, which made the task more challenging. If we had been allowed to use JSON files, we could have used the GSON library for automating serialisation and deserialisation instead. Moreover, we had to design many new user interface components to facilitate this, such as windows to create, edit and delete airports, runways and obstacles. We then iterated on this to use AES encryption to store these files securely, such that it would not be possible for a malicious third party to tamper with them - this also took significant time, as lots of serialisation and deserialisation code had to be modified to support the encrypted files, along with new UI components to prompt the user for the encryption password when required.

Lastly, producing a cross-platform JAR file also proved difficult with the constraint of only being able to hand in a single JAR file, as JavaFX includes a lot of native code and modern Java programs are designed to be compiled to an executable file separately for each platform. Nonetheless, we managed to produce a JAR file that worked across all of our respective devices, including different operating systems and CPU architectures, which required a lot of collaboration and debugging from all team members.

2.4 Effort estimation methodology

As a team, we utilized Agile Planning Poker, as taught in the lectures, for our effort estimation. This approach was beneficial because it brought collective intelligence into play, which is often more accurate than individual estimations. Planning Poker definitely helped us understand the task scope better. The discussions that arose from differing estimates allowed team members to express their understanding of the task, its risks and its complexities. This shared understanding allowed us to arrive at more accurate estimations as a group.

There was a noticeable improvement in our estimations throughout the project. Our knowledge about the project increased, and our estimations became more accurate as we progressed. In addition, the team became more cohesive, and we better understood each other's skills and work patterns. This familiarity made our Planning Poker sessions more efficient and our estimations more reliable.

2.5 Workload balancing

We balanced the workload by assigning tasks to team members based on their skills, experience, and availability. We held regular meetings to discuss progress and reprioritise or reassign tasks when needed. To organise our time more efficiently, we could have set more specific deadlines for individual tasks and used project management tools to track progress and identify bottlenecks earlier in the process.

When allocating tasks at the beginning of a sprint, we estimated and put into consideration how long each of the tasks we had identified. Unfortunately, some of these assessments proved inaccurate and resulted in some mishaps that affected our progression. For example, in the first increment, we underestimated how long it would take to get the basic framework of the tool operational, such that we could begin work on the user stories allocated to that sprint. It was difficult for multiple team members to start work on the code without the basics, such as the build script and a JavaFX window set-up, so some progress was blocked until that was completed. As such, several tasks were reprioritised from the first increment to the second when it became apparent that their completion in time for that deadline was impossible.

We acknowledged that different members of the team had different skill sets, strengths and weaknesses, and we used it to our advantage. For example, some members were very experienced with tools such as Figma and LucidChart, so their time was prioritised on creating design artifacts for the report, such as storyboards, mockups and UML diagrams. Other team members had stronger programming skills than others, so it was sensible for more complex programming tasks to be assigned to them.

3 Tools and Communication

Throughout the project, we utilised various tools that played essential roles in software engineering, collaboration, and communication. Let's discuss each category and evaluate their respective value, highlighting the most effective ones:

- **Integrated Development Environment (IDE):** We all used IntelliJ IDEA as our primary IDE, which provided a comprehensive set of features for coding and debugging in Java, such as the built-in JavaFX

SceneBuilder. IntelliJ IDEA is the IDE taught in the Programming 1 and 2 modules, so all of us already had lots of experience with it.

- **Version Control System (VCS):** Git, along with GitHub, was crucial for collaborative development. It allowed us to track code changes, manage branches, and merge code efficiently. The VCS enhanced code collaboration, enabling multiple developers to work on different features simultaneously while maintaining a coherent codebase.
- **Testing:** For automated unit testing, we used JUnit, as some of the team already had experience with it, and it is the most popular unit testing tool for Java. We were able to combine unit testing with our Github actions CI/CD pipeline to facilitate automated testing, ensuring the reliability and stability of our code upon every commit. We were able to identify and fix issues promptly and maintain the overall quality of the software. We also made extensive use of scenario testing: for each user story, we wrote acceptance criteria and then carried out each stage of the user story, marking it as either pass or fail, so that we had confirmation after all code was completed that we had fulfilled all user stories, without any issues impacting them.
- **Design Artifact Creation:** We used LucidChart to create our UML activity and class diagrams, as we found it very easy to use, and it has built-in support for the artifacts required to draw UML diagrams. Next, we used Figma to create the storyboards, mockups and burndown charts. Figma is easy to learn but hard to master and requires a lot of practice to use effectively. Fortunately, we had group members who had experience with it, which we used to our advantage to create professional design artifacts for our project. Figma also supports working collaboratively in real-time and sharing designs with other members of your team.
- **Submission Document Creation:** We used LaTeX to create our hand-in documents, making heavy use of its features such as figures, table generation and automated formatting. We all had prior experience with LaTeX, as it is taught in a first-year module, meaning that no time had to be spent learning how to use it by anyone. We used the Overleaf's online LaTeX editor, as it allowed us to collaborate on documents in real time.
- **Project Management:** Notion was used for project management, task tracking, and progress monitoring. It allowed us to create and assign tasks, set priorities, track the status of work items, and visualize the project's overall progress. These tools were valuable in keeping the team organized and ensuring everyone stayed on track.
- **Instant Messaging:** We utilised messaging platforms Microsoft Teams and Discord for quick communication, sharing updates, and addressing urgent matters between team members and our supervisor. These platforms provided real-time messaging, file sharing, and integration with other tools, facilitating efficient team communication. In addition, we frequently utilised Discord's "Events" feature when scheduling meetings.
- **Video Conferencing:** Microsoft Teams was used for virtual meetings, including stand-ups, sprint planning, reviews, and working sessions. Microsoft Teams enabled us to have face-to-face communication, screen sharing, and effective collaboration, especially over Easter, when team members were working remotely.
- **Evaluation of Physical Meetings Strategy:** Our team followed the Agile framework, and physical meetings played a vital role in facilitating collaboration and progress tracking. The frequency of physical meetings depended on the project's timeline, team availability, and the nature of the work. Daily stand-ups were conducted every working day, while sprint planning and review meetings occurred once per sprint. Working meetings were scheduled as needed. Here is an evaluation of our meeting strategy:
  - **Daily Scrums**: We conducted daily stand-up meetings, either physically or virtually, to discuss individual progress and potential blockers, and coordinate tasks. These short meetings ensured transparency, aligned team members, and identified any impediments early on.
  - **Sprint Planning**: Sprint planning sessions were held at the beginning of each iteration. The team discussed and estimated user stories, set sprint goals, and defined the scope of work for the upcoming sprint. These sessions helped us establish a shared understanding of the tasks and priorities for the iteration.
  - **Sprint Reviews**: At the end of each sprint, we organised physical or virtual meetings to demonstrate completed work and gather feedback from stakeholders. These reviews provided an opportunity to assess the increment, validate requirements, and make necessary adjustments.

  - **Working Meetings**: As needed, we scheduled working meetings to collaborate on specific tasks or resolve complex issues. These meetings allowed team members to work together closely, share knowledge, and address challenges collectively.

4 Advice to Next Year's Students

- Effective Communication Strategy:
  - **Set up a consistent channel for communication**, such as a Discord server. Encourage team members to use these channels for sharing updates, asking questions, and seeking clarification.
  - Emphasize the importance **regular and active communication** within the team. Encourage prompt responses to messages and facilitate open discussions to foster collaboration and maintain transparency. Our team members were often available, allowing for easier brainstorming sessions or to ask a quick question.
  - Encourage the team to **document code, important decisions, meeting summaries, and project milestones.** This will help maintain a shared knowledge base and allow team members to reference information easily. Often team members were confused by code written by another member due to the lack of documentation, which was solved as soon as everyone started to document their code properly.
- Commitment to Agile Working:
  - **Understand the importance of Agile values**, such as collaboration, adaptability, and iterative development. Learn to understand and appreciate the benefits of Agile methodologies in delivering high-quality software. Utilising Agile values allowed our team to
  - **Regularly review and reflect** on your progress, evaluate your processes, and make necessary adjustments. This will help your team to continuously improve your work and adapt to changing project requirements.
- Time and Task Management:
  - **Set clear goals and priorities** for each iteration or sprint. This will help you stay focused and ensure that important tasks are completed within the given time frame.
  - **Break down larger tasks** into smaller, manageable units and estimate the effort required for each. This will aid in better planning, tracking progress, and meeting deadlines.
  - **Try not to underestimate time lengths,** and instead overestimate the effort for each task on time, as this will help ensure you finish your task on time. We found that we initially significantly underestimated the time required for many tasks. You are unlikely to work wholly on a single task without any breaks or distractions, especially as a university student with other responsibilities. Furthermore, if you encounter issues that you had not foreseen, or simply need to spend some time learning things, the time required for each task can grow very quickly.
- Collaboration and Teamwork:
  - Emphasise the **value of teamwork and collaboration**. Encourage each other to actively support and assist their teammates, share knowledge, and celebrate achievements together.
  - **Leverage the diverse skills and perspectives** within the team. This will contribute to a richer problem-solving process and lead to innovative solutions. Our team highly valued the unique skill sets possessed by each member; those proficient in Java took charge of programming tasks, while those with artistic abilities assumed responsibility for graphics-related assignments.