Cohort 2 Team 1 (Assessment 1)
Ahmet Abdulhamit
Zoey Ahmed
Tomisin Bankole
Alanah Bell
Sasha Heer
Oscar Meadowcroft
Alric Thilak

Cohort 2 Team 2 (Assessment 2)
Bader Albeadeeni
Dan Hemsley
Jennifer Bryant
Mathilde Couturier-Dale
Oliver Elliott
Rosie-Mae Connolly
William Mutch

Risk Management Process:

The team used a lightweight 4-step risk management process as the first version of the project is small scale. This meant a simple, iterative review cycle kept the process efficient without any excessive documentation overhead and ensured that risks were continuously identified, assessed and monitored throughout development.

1. **Identification:** Risks were identified during weekly sprint planning meetings through team brainstorming and reviewing the previous sprint issues discovered.
2. **Analysis:** After being identified, each risk was assessed qualitatively using a 1-5 scale for likelihood (5 = likely) and another 1-5 scale for impact severity (5 being the highest).
3. **Planning/Mitigation:** For each risk the team proposed practical actions to take to reduce the likelihood and impact of every risk identified.
4. **Monitoring:** At the end of each sprint, the risk register was reviewed and the impact/likelihood scores were altered depending on situation changes. Each risk was also assigned an owner responsible for tracking it and updating its status during reviews.

Risk Management Extension

As the project was inherited from a previous team, the risk management process was extended to account for additional risks introduced with the transition. A dedicated period was allocated at the beginning of the development to reviewing, understanding, and documenting the inherited code and existing documentation in order to identify risks. These risks were recorded by maintaining the same 4-step process and risk table format to ensure consistency across the project. Most of the identified risks related to unfamiliarity with the project as well as potential issues with the original submission.

Risk Register Format Justification:

Formatting: The group used a tabular register to manage risks. The tables are grouped by risk type (technical, organisati

Technical Risk ID:	Risk	Likelihood	Impact and Severity	Mitigation	Ownership
TR1	New tools	Likely	3 - Likely that there will be small bursts of defects and reworks from misconfiguration. New tools could cause early uncertainty and slower delivery.	Coding members begin to practise and 'play' with the new tools from week1. This early familiaration will help ease uncertainty before the main coding begins.	The development team learns new tools. The Scrum Master creates practice sessions and watches progress. Method Leads focus is documentation.

onal, scheduling, quality) to help keep each one concise, informative and easy to maintain throughout the project. Each risk entry records what the risk is, likelihood, impact, mitigation and ownership.

Risk Register:

Likelihood Key:
Likely = 5/5
Possible = 4/5
Unlikely = 3/5
Rare = 2/5
Very Rare = 1/5

Impact Key:
Very High = 5/5
High = 4/5
Moderate = 3/5
Low = 2/5
Very Low = 1/5

TR2	Members do not show up to meetings.	Likely	4 - May delay decisions or cause tasks to become misaligned. This will force	Using Whatsapp as a continuous communication channel. Each member	Scrum Master structures meetings and ensures engagement. Team

			reworks and idle ‘blocked’ time. This would delay tasks and testing time. This may lead to schedule slippage as the live feedback loop can't work with poor coordination and missed touchpoints.	should post the progress and any roadblocks. This should also be daily. This will maintain the coordination even if attendance drops. Tasks won't stall.	members update through WhatsApp and Google Docs. Method Leads check sprints and records patterns in attendance.
TR3	Integration issues	Possible	3 - This could trigger rework cycles and delay the next demo. It is likely that integration problems will convert into re-works (extra time and costs) and also schedule slips.	By adopting integration with small and frequent merges to a shared branch where every merge must compile. If we schedule a weekly integration sprint it will help catch incompatibilities and mean fixes are easier.	Developers manage code merging and testing. Architects make sure UML and design align with the codebase. Scrum Master schedules weekly integration checks(sprints) and creates meetings for issue resolution.
TR4	Learning curve for LibGDX and UML tools	Likely	3 - Slower development of features while learning APIs and modelling notation. A weak UML quality will risk miscommunicatio n, ultimately leading to a rework.	Focus on understanding UML examples online and from lectures and use short internal meetings to discuss and help each other understand. This will improve the overall team's clarity of UML.	Scrum master will make sure members understand UML examples by continuously discussing members' work.

TR5	Performance limitations	Possible	3 - This will force late optimisation or feature cuts. This could result in a worse user experience and missed targets.	If we define a performance baseline early on and keep graphics light with incremental testing (performance) after each feature, we should prevent late surprises.	Scrum master reviews work done after weekly sprints. Method Leads should update the plan accordingly. Members should do work agreed at sprints and communicate delays.
TR6	Version control conflicts	Possible	3 - Time will be lost and potentially stall the team near deadlines.	We should only merge features after code review and resolution. This minimises risks involved with integration.	The scrum master is to maintain the communication channel. Architects should be updating developers and should make sure documents show the changes.
TR7	Inherited codebase complexity	Likely	4 - Unfamiliar or poorly structured code from the first team may slow new implementation with the second team.	The team can review original code and document classes and systems.	Developers analyse the code. Scrum master monitors progress.
TR8	Hidden dependencies	Likely	3 - Undocumented dependencies between classes might cause unexpected behaviour.	Class dependencies and relationships should be reviewed and documented before making changes. May require debugging.	Coders and architecture documentors should review, document, and share any useful information.

Organisational	Risk	Likelihood	Impact and	Mitigation	Ownership

and Team Risk ID:			Severity		
OTR1	People are sick	Likely	4 - Tasks could be delayed or reallocated which will lead to a short-term productivity drop.	We should keep tasks small and share documentation so others can help.	Scrum Master will reallocate work. All members will support key tasks.
OTR2	Communication breakdown	Possible	3 - Misunderstanding could lead to duplicated work (or missed work).	Continuous updates through whatsapp and a clear outline of roles in the documents will help prevent this.	Scrum master will update through WhatsApp. Method Leads will maintain the planning document.
OTR3	Uneven workload	Possible	3 - Some members could be overworked and other may end up idle which will reduce quality and morale.	Weekly sprints provide an opportunity to rebalance tasks.	Scrum Master will rebalance the sprint tasks. Method Leads will track all of the changes and each member should assist as needed.
OTR4	Limited availability	Likely	4 - Less available members will cause sprint goals to not be met.	Continuous communication on whatsapp means we can plan ahead, prioritising core features.	Scrum Master manages scheduling. Team members make absences known early.
OTR5	Lack of knowledge transfer from previous team.	Likely	3 - Missing knowledge about design could cause mistakes due to misunderstandings.	Team will go through code and documentation together and contact the previous team for information if needed.	All members should participate.
OTR6	Inconsistent code practises	Possible	2 - Differences between the first team's code and second team's	Coding standard can be agreed on by the inheriting team based on	Developers can follow the agreed standards.

			code could damage readability and maintainability.	the original code. Exceptions can be made.	

Scheduling and Delivery Risk ID:	Risk	Likelihood	Impact and Severity	Mitigation	Ownership
SDR1	Underestimating time needed to complete task	Possible	4 - Could delay the delivery of sprints. Could lead to features being rushed or even dropped.	It would be a good idea to break tasks into smaller substacks. We could also include buffer times in sprint planning.	Scrum Master will lead the planning of sprints. Method Leads will record any issues with estimation. Developers will provide timing input.
SDR2	Missing sprint goals	Possible	3 - The would build pressure and also reduce visibility of progress.	We should prioritise the minimum viable.	Scrum Master monitors the progress of sprints. Method Leads direct the team in prioritising tasks.
SDR3	Slow familiarisation time	Likely	3 - Could take more time than expected to understand the inherited project, delaying development.	Time for familiarisation should be accounted for when sprint planning.	Members of coding and documentation teams should report progress to the entire group. Scrum master ensures that timing is on track.
SDR4	Defects in inherited code	Possible	3 - Pre-existing bugs may require unplanned fixes which could disrupt development.	Prioritise early testing and address bugs before implementing new features.	Coders can identify and fix bugs in the code.

Quality and Requirement Risk ID:	Risk	Likelihood	Impact and Severity	Mitigation	Ownership

QRR1	Misinterpretation of customer requirements	Possible	4 - Features will not meet their expectations. This would also waste the developing effort.	We should confirm requirements early with team reviews and ensure clear meeting notes.	Zoey, Sasha and Ahmet should confirm the requirements. Scrum Master should review the requirements.
QRR2	Lack of traceability	Possible	3 - This would make it hard to verify tasks which fit the requirements.	Git commits and continuous updates will be crucial.	Developers will ensure documentation. Developers keep the commits linked. Scrum master checks progress.
QRR3	Lack of testing	Possible	4 - Could lead to undetected bugs or unstable features. This will reduce the quality of features.	We should implement unit tests and run playtests after each sprint.	Developers handle the testing. Scrum Master monitors the inclusion of tests.
QRR4	Inherited project not meeting current requirements	Possible	4 - Inherited project might not meet all requirements which will take time to go back and implement before continuing with further development.	Requirements should be reviewed so that gaps can be documented and prioritised.	All members should bring light to gaps in their designated areas so that they can be prioritised.