SCHOOL OF
COMPUTER SCIENCE

CIS3760 - Project Post-Mortem

Group 5
A.Blankenstein, H.Mostafa, J.Subang, J. Phung, S. Narayana

Table of Contents

---

1. What did the team do well?

Throughout the lifecycle of this project, our team communicated amongst each other effectively. This includes team members communicating their meeting time preferences, their updated unavailability for a scrum meeting, bringing up any conflicts that should be addressed, requesting help for a user story they're working on and so on. With a strong communication foundation, the team was able to maintain a collaborative environment, built on trust and constructive criticism. Team members would provide their insight on a feature without bringing down another team member's work, while suggesting enhancements that would contribute to the project's success. With a common understanding, we were able to resolve conflicts in a professional manner. This also contributed to a good cross-functional collaboration within since members of our team had a wide variety of skill sets. At the very beginning, Alex and Sanjee helped out with their experience in Android Studio, and transferring such knowledge to the team. Jarryl led a Git crash course, familiarizing the team with Git commands and merge requests. As for Gitlab onboarding, Habiba introduced the team to the project management features of this tool. Finally, Jason contributed to wireframing and prototyping, learning more about Figma and UI/UX.

Another area our team excelled at was documentation. We utilized the available Gitlab tools and commands to provide thorough documentation on the Wiki which allowed us to better stay on track. Team members were able to provide descriptive commits and link these commits to their corresponding issues, along with screenshots of functionality and testing. This helped keep everything organized and served as a means of accountability. As for technical documentation, we utilized individual branches to track individual progress and had proper version control of the application. This allowed everyone to work on their own issue without breaking any of the existing code base. To support that further, JUnit testing and System testing were implemented and well documented across all issues. This was further enhanced by the setup of a CI/CD pipeline that confirmed all issues met their definition of done criteria and were ready to ship.

Another practice our team did well is demo preparation and presentation. To avoid any last minute issues, we adopted our own code freeze policy after sprint 1, so that all features would be ready to present and all bugs would be identified early on. We exerted many efforts into our demo content, and ran meetings before to prepare for the presentation element of it. In a real world implication, demos are to be presented to our clients, so we wanted to ensure they're as clear as they can be and detailed enough to explain our thought process and progress over the designated sprint.

2. What issues arose during the project?

Given the remote environment of this course, the onboarding process took a bit longer than it normally would, especially since not all team members knew each other prior to this project. At the very beginning, there were some communication challenges as some team members were not active on the main communication medium chosen, Discord. This is further discussed under section 2.3. After the first sprint's retrospective, we were able to identify communication issues and discuss them openly as a team, coming up with appropriate solutions that led to effective communication throughout the rest of the project. In addition to that, we had minor issues correlated with time management at the very beginning. Members of the team weren't as consistent with scrum meeting times, meeting soft project deadlines and we had a tendency of overestimating and underestimating sprint velocities, until we finally figured out the ideal time estimate. Further discussion of our time estimate challenges is provided under section 2.1.

As we addressed the core issues of communication and time management, we had some minor issues arise relating to more technical aspects of this project. For starters, we unintentionally chose a code base that turned out to be longer than required, and hence had to change to another code base which delayed our first sprint by some time. As we finally found an open source code base that meets the criteria and settled on Minimal To-do, it turned out to be overlapping with Group 6. We hence had to brainstorm more ideas to distinguish our features from theirs, while maintaining the main goals of this project: organization and hierarchical views. We were however able to overcome that challenge and reach a common ground with Group 6, and faced no issues moving forward.

Throughout the project, we ran into some merge conflicts as different user stories were allocated across team members, and upon individual development and pushing code to our own branches, merging overlapping features sometimes threw warnings, incompatibility errors and variable mismatch occurrences. To overcome this issue, we implemented a code freeze period to ensure we set aside enough time before the end of our sprints to resolve any conflicts manually, and we agreed on common practices to be adopted across the team to avoid incompatible resources and SDK version issues.

2.1. How well the team estimated its Sprint velocity each Sprint? Did this improve over time?

Throughout the span of this project, we learned a lot about how to better estimate the team velocity and allocate time accurately. The first sprint, we had a team velocity of 20 hours making it 4 hours of work per team member. Since it was a one week sprint, we estimated that time but didn't track it as much since we were still getting familiar with the whole process.

The second sprint had an estimated team velocity of 25 hours and an actual team velocity of 23 hours or 4.6 hours of work per team member. This week we overestimated on our estimated velocity by 2 hours. The third sprint found an estimated velocity of 34 hours and an actual team velocity of 40 hours or 8 hours of work per team member. Here we found ourselves underestimating by 6 hours which isn't ideal. The reason for this is

that we overestimated during sprint 2, so we tried estimating a bit less this time to be more accurate. We also didn't account for the chance of running into technical issues during development. The fourth and final sprint had an estimated team velocity of 40 hours and an actual team velocity of 40 hours or 8 hours per team member. This is perfect as we met our velocity target right on the mark.

Overall it is hard to say if we improved over time. More data might be needed with future sprint development targets. With the data we have, we suggest that the estimated team velocity did in fact improve over time seeing as the first few sprints had accurate time estimates, but the final one was exactly what we estimated. Meaning we learned from past experience and finally got it at the end.

2.2. How well did the team coordinate project tasks across the team?

Having a balanced team with members of various expertise allowed us to conquer all the difficulties we as a team faced during the project as a whole. The fact that we all had strengths in a vast array of areas meant that we could very evenly distribute tasks to each member during every sprint. The reason this helped was that the tasks we had during each sprint were vertically sliced components of our application. By doing so we are able to ensure each member of this team worked on something that accompanied their individual strengths very well.

All in all our team was able to coordinate effectively in making sure everyone had a balanced workload when compared to everyone else on the team. When one individual was having issues, others always came to their aid. Because of that we were able to uniformly complete our tasks throughout the project and ensure the work we complete is done so at the best of our abilities.

2.3. How effective were the team communication tools that were used?

Communication is very important for our team and having quality communication tools was essential for coordinating the work for this project. We mainly used Zoom and Discord which was all we needed. A private Discord server was created and only group members were invited to join. Within the Discord server, separate channels were created to organize the group, this included the general, contact-info, meetings, and Android channels. Within the channels, important messages were pinned so that they could be easily accessed by group members by checking the "pinned messages" feature of a channel instead of scrolling through the chat log. Group meetings were mainly conducted through Discord except for Wednesdays, where we would meet in the lab's breakout room. For meetings in Discord, we would use the voice chat and screen-share tools. Screen-sharing was particularly effective because one member could show the group their progress on the application by screen sharing the Android emulator window.

We did have a few challenges in communication at the beginning and this was due to group members not getting a notification when they were messaged or pinged on Discord. First we decided that we would contact the member using text (sms) messaging if they did not reply to the Discord message within a reasonable time. This issue was eventually fixed when notifications were properly set up for all group members so they can see if they are being contacted when they are not online. Once notifications were set up, all group members responded quickly any time they were messaged, and individuals were further encouraged to ask for help and to provide technical assistance when requested.

3. What could be done differently to improve project planning and management in a future project?

For a better overall experience, project planning could be improved by allocating more time for the team to understand the given or chosen code base in depth, and acquiring any prerequisite skills as needed. This would have facilitated the backlog creation process as the team would have a better idea of the technical implications of most user stories, and would estimate time accordingly, leading to more accurate velocity estimates. Moreover, it would be ideal if we could list the possible improvements by investigating the code base and its functionality, rather than roughly listing possibilities by just looking at the front-end side of the mobile application at hand. Additionally, given some limitations of open source projects, it could have been easier for us to work on a mobile application from scratch, where we followed our own practices, implemented our own design patterns and had a better understanding of our code throughout the project planning and management processes. As opposed to having to allocate time and resources into understanding poorly documented parts of the existing code base, along with constantly refactoring parts of it. Another technical improvement consists of following a test driven development (TDD) approach early onto the project to set the project standards, improve productivity and identify bugs early on. Having team members adapt to test-driven development, learn JUnit and set up CI/CD pipelines early on would certainly contribute to an improved, well planned project experience.

A more non-technical area of improvement consists of building a stronger team morale and getting to know one another before committing to such a long term project. Starting with icebreakers and thorough introductions, and allocating some time for team building activities could contribute to a more successful team with more effective communication. In addition to that, ensuring communication means are properly set up prior to commencing the project is ideal as it would save the team time and efforts. This includes having backup contact methods in place to make sure individuals can be contacted even if they are away from their workspace, in early stages of the project management process.