Overall Project Retrospective

Note: All retrospectives, including for sprint 5, can be found in the compiled report

Positive lessons learned:

- A rigorous meeting schedule helps immensely in achieving a streamlined workflow, as well as to allow for constant collaboration between members. Sprints with fewer meetings had far worse performance.

- Comprehensive meeting minutes are essential to keep absent teammates on track and aware of constant changes. Whenever we slacked off on this task, we could notice a dip in productivity, and small errors/misunderstandings occurred.

- Learned about git and github practices such as keeping the main branch protected, committing frequently, making atomic commits, using feature branches, and linking github issues in commit messages.

- Meetings with the TA were pretty crucial, and we should have booked some more meetings. These meetings were helpful for better understanding project requirements, and helped us know whether we were progressing at a good pace.

Negative lessons learned:

- For many, this was a first development project involving a larger (5+) team count. A different work methodology was required. We progressively learned how to appropriately communicate, divide tasks, and track task completion. In retrospect, we could have better utilized tools that would have enhanced communication and organization such as Jira.

- The architecture is super important in a big project, it should be discussed, created and approved by all members then we discuss splitting the tasks.

- Assuming that code quality can be maintained by each person's skill is unwise and implementing tools like SonarQube early is a much better practice.

- Setting global variables and possibly hosting from an earlier phase of the project would have been better to have fewer links and code to change. Doing it so late made the transition harder due to the higher number of pages and features implemented at that point.

As the semester-long project comes to an end, many things were learned in the process. Not only technical skills, but also many interpersonal and personal ones. We grew over the semester as a team, and although some dynamics/ideas came organically, some had to be enforced. Either way, we learned from it all. At times it was difficult, but we came out more competent and with many new insights on how to approach our upcoming Capstone projects. EstateFlow taught us many things and although it has many flaws, we can at least look back now, notice what they are, and explore ways to avoid/manage them. As opposed to the beginning of this project, where many of our biggest challenges were not even expected at all, and for those we expected, our approaches for facing them could have been significantly improved.

On the positive end, we learned:

- Early on, we decided that since there were no tutorials, labs, or lectures, we had to make time for cooperative work ourselves. Without these enforced blocks of time, we also knew each member had more free time for these meetings. In light of this, we found two times in the week where most of our members were free and started bi-weekly meetings. Early in our sprints, we respected this schedule and saw good results which seemed obvious, but we also slipped up at times and that served to teach us the alternative. Whenever this happened, there was markedly more confusion and discord within the team. Thus, we learned both the benefits of following this rule, and hindrances that come along with not.

- Although we picked popular times within our team, the team itself has 10 members and thus not everyone could attend every meeting. In light of this, we started to keep meeting minutes to keep non-attending members on track and aware of major decisions and talking points. Thus, they would be able to make comments, or share ideas regarding what we discussed at a later date. Similarly to the point above about meetings, not respecting this rule led to issues in productivity, efficiency, and most importantly more errors.

- For many in our team, this was our largest scale project to date and this was an extremely good opportunity to practice a fundamental aspect of modern software development: Git and Github. We learned and enforced many ideas such as protecting the main branch, frequent and atomic commits, github issues and linking said issues to commits. As an invaluable tool in the job market, as well as for personal projects, learning to make the most of it was a great lesson.

- Sprint meetings and meetings with our stakeholders (TA and professor) were extremely important. As students new to this type of large scale project, we had

many assumptions and ideas, but with the help of meetings, advice, and the detailed grading rubrics throughout the sprints, we were able to refine our project significantly. This is both in terms of technical skills, but also qualitative decisions such as what features to bring to mobile. This has taught us the importance of both consulting with our stakeholders more often to get a better idea of their vision, but also that consulting with senior developers is quite useful in forming a better project.

Through this project, we also learned many “negative” lessons. These were born from mistakes and assumptions on our part but in the end, any lesson is useful and highlighting how to address them is a very important part of the learning process.

- As stated above, this was the largest scale most of us had ever worked at and it led to a very simple problem: miscommunication and task confusion. We created a Discord server and organized it early as many of us had done for other projects, and relied on that for communication. This was thought to be sufficient since it was how most of us had approached projects in the past but obviously in retrospect, it was not enough. Over time, we learned to organize ourselves better and how best to divide tasks, track tasks, etc… However, as we learned organically over time, we ran into mistakes such as people working on the same task, or members waiting on perceived prerequisites when in reality they were already done. Learning from our mistakes refined our group’s work methodology but it could have been solved much more early.

  Implementing tools such as Jira and/or task tracking program tools would have been extremely beneficial. We cannot say we would not have made any mistakes at all, but I am sure we would have had fewer of them and improved workflow efficiency.

- We mostly focused on our software architecture in the first sprint but the team engagement in this process petered out as the sprints went on. Not to say it was ignored, but focus was shifted too heavily on implementation/development. Putting more time and attention to the architecture would have been quite beneficial in the long run as we had to do some major refactoring work in later sprints. Refactoring is a common occurrence in projects, but the scale of it can be reduced and improved with better planning. Thus, we should have involved more members and ideas in our design phase.

The simple solution would have been to have more robust discussions and focus on this aspect of the project to minimize future work and time lost to delays due to that work.

- Early on, we did not make a point to set hard rules regarding code management conventions. We simply assumed that as students, we are constantly immersed in projects that encourage good coding practices, and thus we would automatically respect such practices/conventions. In reality, that did not happen as easily as assumed. Conventions were applied inconsistently and even if a developer was great at respecting a few of the rules, they may not have respected others. As the project progressed, we felt pressed for time and thus did not want to work backwards to fix our mistake, which led to a compounding issue. It takes much longer to fix a mistake, or add missing information, than to simply do it right the first time by being diligent. This is a common problem to have in a large scale project, but definitely a valuable lesson to learn early in this context where it is safer.

  To solve this, we definitely need to set better rules and encourage following code conventions, but we can also implement tools such as SonarQube to help enforce code quality. Doing so earlier would have improved code quality in the long run and reduced the chances of many of our errors/bugs appearing as the sprints went along.

- We implemented web hosting in sprint 4 and although it did not take long to host, the original version hosted was riddled with issues. This was due to not using global variables consistently, which led to more code/links having to be changed to host all our pages correctly. Local hosting makes development more seamless to be sure, but in the long run, the transition should have been done earlier. Doing it so late made it harder due to the higher number of pages and features implemented at that point, and thus more time taken to transition, as well as more possible errors or problems missed.

  Hosting the website earlier, possibly from the second sprint, and employing global variables more wisely would have been beneficial to the group workflow and allowed for the time used to solve this issue to be used more productively.

To conclude, this project was an invaluable opportunity to learn the hurdles and challenges of working on the type/scale of projects that are typical in the job market, in a safe, controlled environment. We were able to learn lessons through good and bad decisions, but in the end, all lessons improved us as developers. This has certainly given us a good introduction to the subject, but I am sure that the full Capstone project will be an extremely important exercise in our formative training to become competent developers, team members, and collaborators.