ChocAn Project Report

Authors:

Daniel Mendez

Alex Salazar

Arman Alauizadeh

Kyle Zalewski

Alex DuPree

Dominique Moore

12.06.2019
CS300 Fall 2019

INTRODUCTION

The following report consists of project retrospectives from each member in our group. Each retrospective contains the authors feedback on what went well, what could've improved, as well as a discussion of lessons learned.

Alex Salazar

What Went well:

I believe our documentation of the project went well. While we were marked down a few points for assignments, overall I was I was happy with the grades. For each document, every member was asked which section they would like to work on to ensure everyone got was able to do what they wanted. If they did not care, they were just randomly assigned. A slack channel was created so that if anyone ever needed any help, on the documentation or coding, someone was there to answer questions.

What Didn't Go Well:

Early communication on the project code did not go well and resulted in uneven work distribution between team members. We did not meet up as a group and talk about the design and code early in the building process. Because of this, what ended up happening is a member or two took charge of the design and started coding before the rest of the team members. The rest of the team was fine with this and just asked to be assigned what code to do, which seemed fine. However, it was expected that everyone was going to be keeping up via the slack channel and github, but that was not the case. This caused confusion once others started to participate because they did keep up or ask questions early on when they should have. All of this with the fact that the code was a little more advanced than expected made it difficult to contribute. For future projects, I would recommend if the team plans to do remote type work, they should do a few initial meet ups just to build the foundation together.

Lessons Learned:

I learned that the team needs to fully communicate and meet up early on rather than later. As a group we should ensure that everyone is keeping up with what is going on so there is no confusion when they start trying to contribute. However, someone will always take charge and it is the personal responsibility of each member to ensure they

are keeping up with all the work.

Alex Dupree

What Went Well:

Because the unit testing was expansive, whenever code was changed, we were able to quickly spot and find regressions because of failing unit tests. The design and architecture of the code was also very modular, which allowed us to develop components separately and conduct ambitious refactors without breaking the application. All the tests were automated and continuous integration was hooked into the pull request process, which allowed us to debug and verify new application features before they were merged. The GitHub repository was also hooked into the group Slack channel which allowed us to communicate early and often about the exact changes that were happening in the codebase. We also used the project boards and issues feature of GitHub to highlight exactly what work still needed to be done in the project, and open bug reports.

What Didn't Go Well:

We were unable to sync up as a group until the last couple weeks because of conflicting schedules and other priorities. Most pull requests had to be merged because they were open for days with no feedback. As such, the code base evolved with only a minimum amount of interaction, which created a general climate of confusion about the current state of the project. I think the culprit for the communication deficiency was assumptions. It was assumed that the Slack and GitHub were being used and checked often, and also that we all understood how to navigate a repository on GitHub. Instead, regular face-to-face meetups and group coding events should have been conducted in lieu of our remote type of workflow we had going on. This would ensure that everyone is on the same page and have a clear understanding of project expectations.

Lessons Learned:

I learned a lot about developer tools and how they can make the software development process much easier. Using version control with git, and hosting the repository on GitHub made collaborating a more sane process. Continuous Integration was extremely useful in ensuring the application was always buildable and well tested. For future projects I think it will be critical to meet regularly as a group to have better channels of communication, ensure that everyone understands what tools are available, and encourage feedback.

Arman Alavizadeh

What Went well:

Communication went well. Everyone did their part of the assignments and there were very few, if not non-existent, conflicts within the team. We challenged ourselves with how we approached the application and it was a new experience for me as a programmer. I learned a lot by working on the project about abstraction between application layers, application states, unit testing and different types of testing and more. I also learned a lot about github: merging, pull requests, version control, committing, etc. Learning how to write unit tests, coordinating with others on writing features for the application, it gave me a better grasp of what the industry is like. Our communication over slack was good as well, we were always willing to help each other out in and out of class.

What Didn't Go Well:

Most of us were confused as to the state of the project. The infrastructure of our codebase was mostly made by two of our group members and the rest of us didn't contribute much to that infrastructure. The other group members implemented the features of the application such as creating accounts, viewing them, updating, etc. Because most of us didn't work on the framework, it was a bit difficult to start work on the project. We also didn't communicate much during the early portions of the project, which is probably why this ended up being the case.

Lessons Learned:

I learned a lot about abstract application layers, github, groups software development, and a lot of different new concepts that were used in the creation of our project. I think if we had more meetup times as a group, then we wouldn't have had so many questions about how to contribute to the codebase and instead just do it. Things like continuous integration also make life as a programmer much easier.

Daniel Mendez

What Went well:

Overall the entire project went well. The repo was well managed and team members were available and willing to work together. Team members were flexible with taking on tasks and assisting in areas otherwise not explicitly their responsibility, which made for a better working environment. Also, the code was tested with every build which mitigated regressions and gave team members confidence in their code. Lastly the group was well managed by our techlead Alex Dupree. Alex wrote our build script, our database, and an in depth contributing guide for our repo to make collaborating on this project much simpler. He also showed some of us how to write unit tests, demonstrated how to implement design pattern effectively, incorporated integration testing into our repo, and was available to assist anytime with understanding the code base. Ultimately we were applying as many programming and software development concepts as we could to make a product we could all be proud of, and I feel we accomplished this.

What Didn't Go Well:

There were a couple of occasions of updates getting pushed straight to the master branch by team members, as opposed to pushing to their respective branch and getting merged to master through the vastly preferred pull request process. Perhaps more upfront training on git and GitHub commands and features could have prevented this. Merge conflicts were a more persistent issue however, since sometimes team members would restructure the code base to fit their current needs without first considering and or communicating the potential impact it may have on other modules in development. This would sometimes force team members to refactor large portions of their code at a time when they thought it was ready. As a team member who was guilty of this I can attest to the importance of communication when refactoring portions of shared code. While conflicts with merges are common, major conflicts are mitigatable if you discuss your design with team members in advance instead of having a technical argument at the time of merge to determine who gets to keep their functionality and who has to refactor.

Lessons Learned:

This project taught me and everyone else how difficult working with others to develop complex systems can be, even when its with people you like. Ensuring that all

team members clearly understand what design principles are being used, why they are being used, and how they can best assist in the development process, is as critical as it is difficult. On a related note, I learned how to effectively implement the visitor pattern, and how to use design patterns to make the code more understandable in general. Next time I have a project the first thing i'll do after breaking down the problem is look for opportunities to apply a design pattern. Then I'll choose a unit testing framework for the language im using, and add circleci into the GitHub repo to handle integration testing. After that I would write some tests, since even a partially test driven development process is better than writing all the tests at the end, and if appropriate I'll write a contributing guide for my repo. None of which I would have known how to do if not for this project and the amazing people who I worked with on it. The primary lesson I learned here is that working with others will typically be challenging, yet also very rewarding.

Dominique Moore

What Went well:

The communication between the team members was surprisingly good. We didn't meet together as a whole group outside of class too often, but everyone was active on the group slack channel. Whenever anyone had a question the other group members would respond as soon as they were able. We worked well as a team, everyone completed their assigned tasks by the deadline and there was only one person delegated to turn in all of the documents meaning we didn't have multiple submissions for our group. A lot of the group members, including myself, didn't understand the environment we were working in, but I believe we managed to learn what we needed to accomplish our tasks. Working in an environment that's completely foreign to me felt like good practice for the real world.

What Didn't Go Well:

I didn't take the initiative on working on the code portion of the project. There were issues that I wanted to tackle, but I didn't know how I could do them in the style we were using. Instead of saying that I would work on a part of the code, I took my time to try and get a better understanding of what was going on. Too much time had passed before I felt that I had enough information to start, but by that time our program had advanced really far. The moral of the story is to not take too long to try and chase your dreams, or else you'll run the risk of it being out of reach.

Lessons Learned:

Using source control was a big help for this project. Git was something that I heard of before, and I tried to learn a little bit of it before the groups were formed. Luckily, I gained a lot of experience with using Git, but I still have more to learn. Another thing that I learned near the end of the project were the basics of bash scripting. I tried to learn it so that I could fix a potential bug, but it turned out the solution I had was incorrect. In future projects I'll strive to be more proactive and take the initiative. I'll try to learn more about the tools that exists

in the real world, and try to implement them so that I won’t be taken by surprised too much when working with others whom have more experience than I.

Kyle Zalewski

What Went well:

Communication was well established from the beginning with the immediate creation of a Slack workspace to divide work and discuss strategies. The workspace, particularly the general channel, was very active over the course of the entire term, and everyone had a pretty clear idea of what they needed to be working on as it came time to attack different aspects of this project. Google Docs for all of the reports proved to be a very helpful tool and allowed everyone to contribute to their sections at their own pace and feedback was very easy to annotate if needed.

The frameworks chosen for testing and building the software ended up being extremely useful for creating modular, well-tested code. The implementation of the base functions of the software became trivial once all of the frameworks were included, and the end result was software that is extremely modular and maintainable. Using Github to assign and track the various tasks was very helpful. There was always a pretty clear idea of what was left to be done.

What Didn't Go Well:

As the term moved forward some ambiguity began to appear in the exact jobs each person was assigned. This was a result of massive schedule conflict between team members. We really only met in person as a team a couple of times during the term. Luckily, Slack communication "picked up the slack" fairly effectively. In a perfect world, a few more whole-team meetings throughout the term would have been helpful in establishing roles and keeping up with schedules. Some members had heavy coursework this term which had the result of running us closer to deadlines than ideal. Having these early meetings would give the team a chance to collaborate on term-long goals and jobs while everyone's attention is guaranteed present, which is certainly no guarantee with Slack.

Lessons Learned:

Early communication and planning can be much more important than it seems. Being computer science students, we mainly want to just get down to coding, but early planning can prevent a lot of issues down the road.

The collaborative tools that Github provides are practically indispensable in a

modern software development challenge. Being able to branch features and require a thoughtful merge back to the master prevents changes from breaking the entire system and allows many people to collaborate on a single system at once. There is still lots of attention required to make it work without issues (something we discovered a few different times with branches merging in strange ways), but the capabilities of the tool are very useful.

Build tools such as premake and testing frameworks such as Catch2 are hugely powerful. They have a learning curve like anything in development, but they provide a modularity and scalability to software that would be extremely difficult to match from scratch. I will certainly be incorporating tools like this in future projects.