Name: Kodie Artmayer

Term: Fall 2022

Previous Team Projects

In this program so far, the only other project that I have had to do that required working as a programming team is in CS340 which I am currently in now. That project requires that we create a web app that allows a user to conduct basic CRUD operations on a back-end database. My role on this project was most of the back-end programming as well as the overall schema of the database, while my partner was to do most of the front-end integration using my SQL code. We ran into a few difficulties while working on this assignment but most of the conflicts were from one person trying to understand what the other person was trying to accomplish with the code they had written, and then trying to use that in a way that fit the assignment requirements and made sense. Both of us work full time and are enrolled in a full-time course load so each of us started cranking out code and not sufficiently commenting on lines or functions that weren't obviously clear. Also, as the size of our project began to grow, having to read through the whole thing and try to figure out what was happening instead of just reading through some comments began to get tedious. A compounding issue also is figuring out a time where both of us can look at the code at the same time and try and explain what is intended and what is wrong in real time. Not making frequent commits also made it difficult to try and go back and fix when something broke.

Working with Continuous Integration

My initial reservation about using a Continuous Integration workflow was each of us working on the same file and then having conflicts that broke parts of someone else's code or having code that was incompatible with each other's contributions. We planned on starting early so that we had plenty of time to fix anything that went wrong and committing frequently so that we could rollback to an earlier commit if something did come up.

The experience with the mandatory Code Review process was overall good. My group members were very responsive, so I didn't have to wait a long time before they reviewed my contribution and left feedback, so I never waited very long to continue working. They also left some good ideas in the comments some of which I acted on and changed and some of which I didn't. Something I tried to make the Code Reviews more effective was to make sure all the checks passed first. Then I would look at the code and try and determine if I could tell exactly what was going on at just a read through, if I was confused at anything I would indicate that in the review comment. I would also see if any of the code could be simplified or if there was a more efficient way to accomplish the task and suggest it. I would also try and make a comment about something I liked or that I thought was clever. As a code reviewee I liked to submit a pull request even though I didn't have my code working to the full specifications of the rubric. I would work on one part and write my tests and code for that portion, then when that was complete and passed the tests, I submitted a pull request to try and merge smaller working portions incase something went wrong when merging, or I had an error I didn't catch, it would be easier for a reviewer to catch a mistake in a smaller amount of code compared to many lines of code.

I think we all tried to write our function using the Test-Driven Development process, however I know I often did not adhere to it exactly and wrote several lines that would pass a few tests I knew were similar, and I think my groupmates did the same as well.

Lessons for the Future

I think working with Continuous Integration facilitates better software development because it forces the developer within certain confines that ensure the code is correct and what is desired from the final product. Setting up the checks and having other people review and integrate the code helps prevent mistakes and helps to keep the code simple and easy to understand. It also makes it easier to roll back and not lose a lot of work if something breaks the code. Mandatory Code Review kind of goes hand in hand with this is a different set of eyes often catches something the original developer looks over. I have had countless times where a code isn't working right and I cannot find the cause after hours of debugging and review, only to take a 10–15-minute break and come back and find it right away. Mandatory Code Review kind of works like that however someone maybe able to catch something you thought would work but doesn't. The person doing the code review also may recognize wasted steps or have a better way of doing something. People have a routine way that they often accomplish things and I think this goes for writing code as well, and sometimes that isn't the most efficient way. Mandatory Code Review forces everyone working on the task to take a step back and evaluate the product as a whole and how it may or may not be the best option. This can help me learn as a developer if I get a review that informs me of a different or more efficient way of doing something I can learn from them.

A very important lesson to take away from this is writing a good extensive test suite when working on a shared code repository. The checks are good for a lot of things but the most important, in my opinion, is checking that all the written tests pass, and that only goes as far as the tests in the file. If someone did not write a solid test suite for their function, then it could pass and merge just fine, but the code probably doesn't work as it was required. Only with a solid test suite can you be sure as a group that when all of the developers have merged their contributions that the code works, and works as it was required to by the project specifications.