Name: Alexander Kim

Term: Winter 2021

Previous Team Projects

While I have worked on many group projects before in other subjects, this is the first time I have worked on a programming team. While I had many reservations this time that I have had on other group projects, such as quality of my team members, availability, etc., I also had some concerns specific to this project. I had not used GitHub in a group setting and I did not have any familiarity with forking or pushing. Additionally, this is my first group project in an online class, so I could not meet my groupmates face to face for discussion or working. To further compound this, I live on the East Coast, whereas both of my partners live on the West Coast – a three-hour time difference. This meant that our ability to discuss and work at the same time was constricted even further. Any problem that arose could have had a significant delay to get a solution because of this.

Working with Continuous Integration

My initial reaction to having to work with continuous integration was that of irritation. To me, who before had not used Git extensively, suddenly being forced to learn about its fork, merge, and other functions in the context of a group project was annoying. Not only that, the need to commit constantly and review each change seemed to be excessive and inconvenient. While I understood on paper why continuous integration is so important in a group programming setting, I felt that it should have been more of a guideline than a requirement.

Once we actually started with the process, I quickly started to understand why it is so important to use a continuous integration approach in practice. Git is much easier to use than I thought and significantly more convenient than sending code to each other via email or other method. The mandatory code review process helped us spot errors or oversights in code that the writer had missed and increase uniformity of the code in general. As a reviewer, I took my time, scouring through each line that had been changed or added, and tried to think how I would approach the problem that was being solved. I then tried to view the problem from the writer's point of view and follow their logic to see why they did what they did if it was different from my approach. I wanted to avoid unilaterally forcing my solutions onto the group, so I tried to be as unbiased as possible with my reviews and only give suggestions where I truly thought there could be some substantial improvements.

As a reviewee, I took a similar approach, in that I did not take criticism to heart and tried to see why the reviewer made any suggestions that they did. Then I would try to rethink the problem, trying to combine our ideas and approaches into a solution. For example, the function that I wrote initially used two while loops to get the desired value. One of my reviewers, however, thought that it was inefficient to use two loops and suggested that I rewrite part of the function with just one. However, the suggestion that he proposed also involved several helper functions that I did not want to implement due to clutter and length of the code. After looking over the problem and thinking about it, I managed to come up with a function that uses only one while loop but does not need the helper functions – in fact, I managed to get rid of one of the helper functions that was already there.

One of the aspects of continuous integration of which I unfortunately never truly got into the habit is the daily commits. In my opinion, it is the most difficult part, especially coming from a past with no group computer science projects, where the closest thing to a commit I had was ctrl+s. Similarly, my teammates also did not commit as regularly as they probably should have. The teamwide failure in this aspect, however, just further highlighted for me how important it is. Each of our initial commits, which were done after we thought we were done with our respective portions, had glaring issues or obvious misses that would have been easily caught if we had committed at each step of our development. Some of the code had major format issues or cluttered and poorly organized logical structure that would have otherwise been immediately addressed had we committed properly.

Despite my initial reservations about using a continuous integration workflow, I think this project helped me grow in many ways. By viewing other people's solutions to problems, getting feedback on my own code, and realizing our failure with the daily commits, my experience and understanding as a developer has increased. I have been exposed to ideas that are different to my own approaches and understand the importance of checking work in for review to catch obvious mistakes. Reviewing and giving suggestions to other people's code helped me grow as a mentor. At first, I was afraid of offending somebody with my suggestions, but I quickly learned to give my suggestions in a constructive, positive manner. Similarly, I think I grew as a team member from getting my code reviewed. Instead of taking offense to suggestions and stubbornly insisting that my methods are the best approach, I learned to look at problems from other approaches and combine the best parts into one solution.

Lessons for the Future

By working through this project, I was able to understand the importance of a continuous integration approach to team programming. By requiring other people to review the code that I wrote, the overall code was improved since other people were able to spot problems that I had missed or solutions that I had overlooked due to tunnel vision. The style of the coding also became much more uniform, which made the software easier to read and navigate.

Similarly, requiring reviews also helped increase our familiarity with programming, helping us become better developers. By reviewing other people's code, I saw solutions to problems that I would have handled in less efficient ways. For example, when converting between string characters and integers, I probably would have used an array containing the characters; my teammates, however, used dictionaries to associate each character with its integer. On the other hand, I was also able to help my teammates reformat and improve their code to increase efficiency and reduce clutter. I helped a team member convert for loops into list comprehension statements instead, which take less space and look neater as well as helped reorder the logic in his program so that there was much less repeated code. These are just some examples of how the review processes help developers improve by learning directly from other people's code or by getting suggestions from code reviews.

Having a solid test suite was important due to the code coming from different people for different purposes. While we each chose a different function to write, meaning that our respective codes interfered with each other's as little as possible, it was still possible for a commit with poorly named helper functions or other problems to affect the results of another function. The test suite also helped catch problems such as edge case inputs or other errors, that would have otherwise gone unnoticed.