Name: Timur Guner

Term: CS 362 Winter 2022

Previous Team Projects

During my time here at Oregon State University, I have done on several projects. I was on team that developed a UI for a bartending app. This revolved around the concepts of usability engineering and creating user friendly interface. My job was to setup the focus groups and gather data to analyze. It began with a paper prototype to present the idea to the consumer about how the app will look. The first round of interviews was a small round robin to gather their input on the design features. Once, we had input from the users, we tweaked the interface based on their feedback and moved on to develop a functional interactive product. When the product was ready for testing, I brought in five individuals and had them separately perform a series of tasks. Each person had different tasks, but all were intended to provide feedback about usability, errors, missing features, and anything else to find all the kinks and issues with the app. I presented my findings to the UI team, and they put together the final version of the product.

Another team I was part of was a team that developed a database application. Our product was based on the needs of a small electronics business to help manage sales and inventory. My portion of the group project was to scope out the needs of business and develop a database with tables that represented their needs. I was the database administrator for the project. On the frontend, I also handled the routing using flask. Flask routes the incoming queries from the front end as a dictionary or list and modify it to a SQL statement to push to the database. My partner handled the front-end aspect with designing the layout and functionality of the front-end tables.

When working in these groups, the main difficulty was time management. We all had our own parts, and each part had a certain deadline in the process. Our own tasks relied on previous tasks from the other team members and vice versa. If a team member did not meet their deadline, then the task of the next person was halted until it was finished. Luckily, both groups had everyone meet their deadlines on time, even it was time crunch. My team members were all diligent and would work extra hours if a deadline was coming close. The other difficulty was setting up weekly meetings. Both projects I worked on had members who worked different hours and were in different time zones. Occasionally, some team members would have to miss the weekly meeting because of these issues. The way around this was communication over discord and email to make sure the members that could not make the meeting would still get all the information that was discussed.

Working with Continuous Integration

When first learning about Continuous Integration, I didn't think it would be a great workflow. I felt that the constant commits, reviews, and approvals, would bog down some of the development as we would have to take time out of our day to do this. However, after looking at examples of this and video demonstrations, it made a lot of sense that continuous integration is a great method. I would not have done our group project another way.

Part of the Continuous Integration was the mandatory code reviews. This was what I was initially hesitant about, as I mentioned before. However, I found this very beneficial. The mandatory review allowed for each team member to review a piece of code that was pulled by another teammate. The review method made sure that no merges were pushed that would overwrite existing code that needs

to stay or merge outdated code. Having the mandatory reviews also gave us a second set of eyes to check if the code meets the criteria and if there is anything that needs to be revised before approved. Along with this, we had an automatic test suite that would fail the requests if there was linting errors or if unit tests failed. Having this automated feature helped speed up the review process because it eliminated some of the work. As a reviewer, I would test the code locally to make sure it worked based on the design by the developer, before approving it. You do not want to accept the pull request without additional testing. When you are a reviewer, it helps you understand how you would develop your code before you commit for reviews. This is because you already have an idea of what the reviewer is looking for and you can better develop a code that will pass the review requirements.

Daily commits were an important part of this project. My team would try to at least commit once a day if possible. Even if there was nothing for me to commit, I would check to make sure that there was no commits and requests waiting for approval. If there were no commits by any team member, we would discuss to make sure that everything was still progressing as expected. With daily commits, code reviews, and approvals, it allowed me to be a better team member because I knew what to work on, what to get done, and when to have it finished.

When I was implementing some of the code, I used Test Driven Development. One of the functions of the project was to develop a function that took the Unix epoch and converted it to a human legible format. Before I began the code, I wrote test units in Python that would be used to make sure the code meets the criteria. Having the tests ready made it easier to design the code as well as faster turn around time. I found that having these requirements already mapped out was key to successful development.

Lessons for the Future

Continuous Integration facilitates better software development because it requires many different factors for excellent teamwork. Each team member can be given a portion of project, develop those features, and provide code to repository. They can each commit certain portions of the code to new branch without committing and updating the main branch by mistake. Each branch can then be merged with updated code to the main repository once approved by another member. This also allows for small tweaks here and there that can be committed daily as separate branches and not necessarily needing immediate pulls.

The mandatory reviews allow the team to never update the main branch without approval. Having this setup is important because it provides a failsafe. This allows someone to become a better developer because an individual will understand what the reviewer of their code is looking for, if that individual has done mandatory reviewing themselves. With solid practice as a reviewer, your code will be better suited to meet the standards when it comes time for your review.

When it comes to Test Driven Development, it can be one of the most crucial techniques to use. I found it useful when I was developing my portion of the project. Having a set of unit tests that tests the requirements of the software, can help engineers develop products that meet the requirements already defined by the unit test criteria. It allows for faster turnaround time and provides a structure base for the development team. Test Driven Development can be implemented in a shared repository as well. This is great strategy to have because code that gets pushed to the repository must meet the minimum requirements already provided. The code could fail the test thus preventing code from being successfully committed.