Name: Derek Gilmartin
Term: Summer 2022

Previous Team Projects
Like many others in the Post-Bacc program, my previous job had nothing to do with computer code or anything to do with the tech field. I spent the better part of 8 years on ships and working in the Intelligence Community as a Naval Surface Warfare Officer so my interaction with development is solely limited to the educational domain. With my background, I have extensive experience working in team settings to accomplish a goal or mission which has prepared me well for software team projects – or as I like to think of it: Team problem solving.

The Computer Science related team projects I've been a part of have mostly focused on a particular learning objective. For example, I created a full stack web app for food composting that interacted with a partners microservice. The learning objective was to focus on Agile development and interacting with other microservices based on a communication contract. My partner's microservice was a graph function that would receive data from my website graph the data and then return the location of the graph on the local machine. The microservice I wrote for my partner was a password generator-converter which communicated on RabbitMQ and would convert a given password to "leet" speak. Another team problem I worked on was a web app that used a MySQL database. My role in the project was defining the queries for our front end as well as writing all the initial SQL to create the various entities/tables in the database. I also worked on a group project for a usability engineering class where we developed a prototype for a cocktail maker app. We had to meet project deadlines for user research, which I was in charge of, prototype iterations, and reports for a pseudo employer (the instructor).

My role in each of these projects was drastically different which has allowed me to develop different skills but the problems are always similar. I've learned in my previous employment that communication is the key to success in almost any problem. Because of this lesson, I always make it a point in my group projects to get everyone onto some sort of comms structure, normally Discord or Teams. This has solved so many problems for my teams because when one team member is struggling with personal or professional problems, the other members of the team are aware (most important) and able to pick up any slack if necessary.

Working with Continuous Integration(CI)
So far in the program, I've been forced to use GitHub for various things like uploading code for a grade check, retrieving problems to solve, or just uploading code for my portfolio. This class is the first to introduce Continuous Integration and I admit I had reservations. While allowing multiple parties to edit the same code seemed like a huge problem solution, I was worried that code would get lost without sequential additions. I also feared that my teammates might break the code that I had written by introducing their own. In the past, I had treated group projects very modularly – everyone completed their own part, and then we assembled it like a machine. Because of my reservations, I intended to check the main/master branch after every merge to make sure it worked and I didn't lose code in the project or test files. As a reviewee, I forced myself to double and triple check my code didn't cause any problems with my team's code. I think this was more important to me for this project because I realized

the possibility for error and I didn't want to be the cause of that error! I later realized that the pull request was already doing this in the build section because of how the .yml file was written – neat!

Code reviews seem extremely useful when you have varying levels of knowledge in coding. Conversely, I feel like code reviews aren't that useful, or rapidly diminish their utility, when you have similar levels of coding prowess. For this reason, I feel like for this project code reviews were more of a check in the block and sanity check rather than a teaching tool. Our team seemed to do an honest review of major pull requests but if a pull request was done to fix comments, or linting errors, the code review seemed tedious rather than helpful.

Working with CI helped me grow in several ways as student and as a future Software Engineer because it showed me how teams can work collaboratively without stepping on each other's toes. I liken this to working in parallel rather than in series. The result is all the lightbulbs, in this case the problems in the problem set, light up quickly and the team problem is solved more easily. The other skill I was able to work on during this project was the ability to see someone else's code, run it through the debugger, and visualize exactly what is happening. This was particularly useful when I was performing code reviews and helped me achieve two things: I went through my partner's code line by line, and I was able to provide recommendations when I saw a better solution.

Lessons for the Future
While I learned a lot from using CI this first time, I took a few key lessons away to make me a better engineer in the future. First, I think CI is extremely useful for development because it allows multiple people to work on a problem simultaneously. Second, Code Reviews are great and should be utilized when the situation calls for them. Third, a thorough test suite should be thought of as an "ounce of prevention" rather than a "pound of cure".

Continuous integration is extremely useful for software development because it allows multiple people to work on a single document at the same time. Similar to google docs allowing teams to complete written projects simultaneously, CI tools enable teams to write code and solve problems much quicker than sequential programming. The lesson I've taken away from this is that whenever possible, use CI for group related projects.

Code reviews are an invaluable part to any team writing code, just like trust but verify is valuable to any life situation. However, I think there should be a distinction between a code review done to teach and a code review done to verify the validity or correctness of a given snippet of code. It would seem that a coding savant would be able to capture any and all teachable moments in a code review without falling victim to the tendency to want to pencil whip (approve) a code review. In future group projects, I think implementing a business rule of when or when not a code review is required would be helpful. For example, if only formatting is changed, no code review is necessary.

A solid test suite is vital when working on a shared code repository. This is for many reasons but chief among them is the ability to verify code still works before a merge. In the case of this group project, the .yml file ran all the tests to verify that none of them failed. If a test failed, the build would fail and the pull request would not be allowed to proceed. This built a lot of confidence in the team because we believed that nothing that passed the build on a pull request, would break the current progress of the main/master branch. In the future, I think Test-Driven Development should go hand in hand with CI if possible.