Harrison Latimer
Group Project: Part 3
CS 362

Team Projects and CI/CD

Since transitioning from working in a manufacturing environment to full time developer, team interactions and dynamics on group work changed tremendously. One primary difference is the level of “agility” in reworking issues with a product. In the manufacturing environment, the process of product development is much slower due to the need to test a product for many different issues that arise. In a software environment though, testing is a priority but the time to develop a product and get it out the door is much quicker because of how easily a bug fix can pushed out. This ability to push fixes to problems nearly instantly makes a group working environment that is more willing to try a wider variety solutions and takes the pressure off the group to send out the “perfect” product. As a team member in a software development group I have been able to test out theories and ideas to solve a problem much faster than the traditional product manufacturing environment.

The way a group can work on a project to rapidly prototype solutions and push bug fixes to customers is through the process of CI/CD (Continuous Integration / Continuous Deployment). This process requires that at the very least some developers on the project review and approve pull requests to master and unit tests all pass. In a perfect world (and with a properly set up project) once code is integrated into the code base that code can be deployed to the user base.

For the group project in CS362, we were not concerned with the deploy aspect so much of CI/CD but we had amply opportunity to work with CI. Since I had the most experience with git and CI/CD I made an initial branch with the three functions for the project stubbed out for each one of use and got this merged into master (once reviews of course). I recommended that we all branch off of master with the stubbed out functions so the group could work on the three functions in parallel. The one issue with this approach is, with developers new to CI/CD, if a branch is merged into master, you need to merge those changes from master into you local branch. What tends to happen is that a developer will try and make a new pull request and there will be merge conflicts due to all the changes added into master from a prior code review and merge. Since this isn’t an intuitive process, I would send out a group message after every merged pull request into master with the commands to get the changes merged from master into everyone’s branch. After a few iterations of the process the team quickly caught on and merge conflicts nearly disappeared. Explaining this process to my team pushed me to understand at a deeper level what git and Github is used for and how to explain it team members in a clear concise way.

What has become more apparent through the process of CI/CD is that barriers need to be put in place to “protect” working pieces of the code base. With developers newer to working on a shared code base the opportunity to accidentally remove or break functioning code is high. By blocking direct merges into master and requiring approval by the team the chances of breaking the project reduces greatly. The second layer of protection CI/CD provides is requiring unit tests to pass in order to allow the process of code review to occur. Running the test suite not only convinces the team that the project still works as it did before changes were made but it also keeps the developer “honest” e.g. proof that code meets requirements. Once a team understands the basics of CI/CD and unit testing the quality of code delivered should improve and be delivered faster.