Davie Truong
dtruong8@ucsc.edu
CMPS 115 7/25/17
Project Reflection Essay

Before taking introduction to software engineering, coding has always been trivial. Coding up to this point has always been accompanied with instructions that guided me through any issues that I would confront. With the great deal of hand holding in my previous coding endeavors, it was a welcoming surprise that coding from an original idea wouldn't be trivial. Although it was initially a daunting task, the material that we learned in class helped to minimize the confusion that would have accompanied us without it. Scrum, being the software development process that we learned, was highly beneficial to the team. It was a valuable tool that got our idea into a working plan. "Moscow" in particular shaped our project into something that was realistically possible to produce in three weeks of development. Although many influential benefits came from scum, there were also issues that arose.

Learning scrum was one of the issues with the class and project because being a summer class, everything was compressed and slightly disorganized. Thus, we were required to employ scrum practices without fully understanding what it was. In our case, we took our idea and applied what we understood was scrum into the first sprint week resulting in a "scrum but". Initially this didn't appear to be a problem because what we did seemed correct at the surface, but what we had was also a "scrum smell". For example, an issue that represents this was the coding style/guideline. We did not create this important document until the middle of sprint 3 resulting in major productivity issues and merging problems. Another problem that arose was that we gave each tasks trivial estimates of when things were suppose to be completed. This created a terrible overestimation on our project workload leading us to over pack our release report and sprint plan early in development. This problem required us to make changes to our final product, and often push our uncompleted work into the following sprint. Another issue in our development process was not making an acceptance criteria for when user stories were done. By not having an acceptance criteria, many of us claimed code was done, and without learning about unit tests until the end of class, we were left with many bugs when code was merged. An example of this in our project was naming conventions and code that was not created to handle all situations. During merging, we had to comb through our code to find the problem, and since the compiler doesn't register mismatched names it took most of our time.

In a group project, usually two things happen, either the team works in an efficient cohesive unit or some dedicated members are left doing a majority of the work while others cling to their coat tails. In previous school projects it had always been the latter, which makes it a welcoming experience when this group project was was the former. I believe this occurred because we went against what was advised and worked with people we already knew. The background knowledge of knowing everyone's strength and weaknesses as well as having the mutual respect for each other made working the project feel professional. Although there were times where we goofed, we knew our responsibilities and that work that had to be done. Since everyone had this mentality working felt like trivial task which resulted in no confrontations.

In the projected everyone but the product owner was required to adopt the role of scrum master for the duration of a sprint. Having this aspect apart of the project gave each person the chance to learn what it was like to manage a development team. In my case, I adopted the role learning that it wasn't required for the most part. After having the weekly sprint plan created, everyone understood what needed to be done. Because of this, there wasn't a need for members to be managed. The scrum board felt like a more effective manager because it gave everyone a task to do when they needed it. However, I do understand that the role would have been more necessary if I had been in a lazy team. Since everyone had to undergo the role of scrum master, I believed that everyone understood the concept of treating other the way you would like to be treated. So there we no instances of power trips throughout the project.

The difficulty of the project came from the lack of knowledge on how to take our idea and make it into a tangible product. In my case in particular, I had never applied my coding to the real world to make a presentable product, thus there was an initial high learning curve. However, since three members in our team already took the mobile applications class, we were able to trivially find our footing by creating a paired programming dynamic. Those that didn't know how to create a mobile application would be paired with those that did thus any questions from the learning processes would be quickly answered. Molding the idea around an android app was also a great design decision because it was the method that had plenty of online resources. Whenever I was unable to understand how to do something a trivial google search yielded the answer.

Overall the project was a great learning experience and simulation on how the real world of software development would be like. The long hours coupled with the painstaking task of finding solutions to unknown problems, made me understand why the career yields a high pay. It wasn't until I worked on implementing the images portion into the app that I felt coding was like running a marathon in which solving one task would require a solution to another task that wasn't considered. For example, since we wanted to add images to the sandwiches for more aesthetically pleasing application, I had to start by figuring out how to apply images. After which I realized that to have a great product, the images shouldn't be hard coded in and should be selected from the gallery or taken by the camera. After having the basic code implemented and being fairly certain it should work, it was frustrating to discover that it couldn't work. The user needed to give the application permission to access the storage and camera before any of the image task could run. So after solving that ordeal, it was further discovered that the method I implemented to apply the images would not synchronize well with the backend code. However, by the time this was apparent, our project was nearing the end and solving that solution would require another sprint to properly implement. Thus it is uncertain how deep the rabbit hole goes.

The endless rabbit hole was what made the project entertaining and worth doing. Starting from a trivial idea, the hole continued to extend beyond what we were able to do. This aspect of the project was what I believe to be the most valuable experience. Learning that a simple idea can become complex and evolve into a great product made the class invaluable. Although I thought our idea was a glorified note book; it presented a more reasonable task for us to undertake in 3 sprints. Had we done anything more complex, I believe we would not have anything to show at the end. For the rabbit hole would not be a trivial task to get out of.