Wyatt Kormick
CSCI 5801
12/13/17

Assignment 9: Post-mortem

The Software Engineering 1 project this semester was to design the requirements and implementation of the dynamic response to urgent maintenance requests system, or DRUMRS. This system, among other things, had to allow for campus community members to submit maintenance requests, for managers of maintenance teams to turn those requests into work orders, schedule and assign those work orders to maintenance staffers, and for those members to mark those work orders as complete. The first stage of the project was writing a system requirement specification (SRS) document. In this stage we wrote out the user requirements of our system, given a list of user needs, individually. We then met with groups and combined our ideas into one SRS document, including conceptual class diagrams, sequence diagrams, requirements, specification, use cases, and tests cases. The other part of the project was to write a design document, detailing system architecture, class diagrams, and class descriptions. As before, this was done individually at first, and later combined with the groups again, along with the addition of nonfunctional requirements, and dynamic models. Overall my group didn't have many problems along the way, and the times that we did, we were held back by them only minimally.

My first attempt at a requirements document, on my own, turned into sort of a disaster. I wasn't quite sure what an individual requirement should look like, much less an entire document of them. As a result, my individual SRS used no template whatsoever; I ended up just writing down a bunch of things that I thought our system should do, and assigned each of them a number. The final product had so many problems: the numbering system made no sense, the requirements lacked any additional documentation, and nothing got explained. This document lacked most, if not all, of the requirements of both a SRS and just requirements in general. This document did not last more than a stage, although some of the requirements actually made it into the final group document.

Being put into a group, especially one with experience in the industry, was probably the best thing that could have happened to help me learn the material. Through them I was able to see what a well-formed, professional looking document looked like. Together we made a requirements document that actually held up through the design phase. Over the course of this project, we only had to make minimal wording changes to our requirements. One example of this is where we changed the word "progress" to "status" in all of our requirements, so that we had a sort of mapping between the "status" field in work orders to the requirements that used that field.

The biggest problem with our SRS document was specification. When it came time to add specification to our document, we felt like we had no idea what specification looked like. In

the end, we split our actual requirements into the ones that we thought were specification, and the ones that weren't, and we left it at that. We soon found out that we had no specification at all, but it we were out of time, we had to move into the design phases. We had to base our design entirely on our requirements and our test cases. While this certainly was a disadvantage when it came to design, I think we all had a good enough idea of what was going on to get a good design down. We didn't get specification done until after our design.

My individual design went a lot better than my individual requirements document. I used a design document template, so I had a good idea of what sections I needed. I was able to explain the design I had in mind, which helped me actually design it. What I think helped even more was my experience. Design is something I have always spent at least some time on while programming. Although I don't often go as in-depth as this project, it seemed kind of easy to come up with the objects and their methods for our system.

As a group, our design went in a bit of a different direction than our individual designs. Our final design implements a "Unit of Work" object. What this allows us to do is retrieve from the data store, all of the lists that our persistent objects are stored in, every time a user accesses the controller. These lists are stored in memory, and all transactions done in the system are done to our local versions of the lists. When the user is done, all of the changed lists are persisted again to the data store.

For a while during the writing of the design document this sort of... different, way of thinking about how the system hindered our progress. Most of us in the group weren't quite sure where this design was going, but we trusted the group member who came up with it. We let him write about the overall design and its justification, while we took the small jobs that didn't require complete knowledge of the design. We wrote most of the class descriptions, we helped make the class diagram, and we made the interaction diagrams. The initial interaction diagrams are where our lack of knowledge about our design really showed. The first sequence diagrams didn't interact with our unit of work at all, persisted objects in completely the wrong way, and were just altogether incorrect. Eventually, after careful reading of our design document, and explaining by the group member, we figured out what we were doing, saw the brilliance in it, and wrote a solid final draft.

Overall, progress on our project hasn't been hampered by problems too much along the way. The problems that really stick out are my own lack of understanding of what the SRS was supposed to look like, which affected my grade, but didn't see its way into the final draft of the document; the complete misunderstanding of what the specification stage required, which was cleared up quickly; and a shaky understanding of what our design actually did, leading to some mistakes in our design document, but these were fixed by the final draft.