David Mednikov

CS361 Summer 2018

Reflective Journal

Tuesday, June 26, 2018

I currently work as a software developer for a digital media company. Before that, I worked for a company that created mortgage loan origination software. My experience has familiarized me with the SDLC and made me comfortable with the process of deploying software SDLC. The SDLC is important for large software systems because large systems can get chaotic or voluminous and the SDLC gives us more control over the deployment of our product. By breaking down a project into steps (Planning -> Analysis -> Design -> Implementation -> Maintenance) we can ensure that every release of our product is sound and stable. Also, the SDLC recognizes that maintenance is important and takes a lot of time, so companies can allocate sufficient resources to a project.

It is important that we write a lot because maintaining documentation is essential. The software will inevitably need to be improved, new developers will work with the code, and users will need to understand how to use it. Documentation serves all of these purposes. I believe that certain parts of the SDLC are product management, but not all of it. For this reason I disagree that it sounds more like product management than software development. Product managers certainly lead the planning and analysis phases, where the problem and requirements must be identified. However, a lead software engineer is responsible for the majority of design and all of implementation and maintenance.

Software engineering allows us to tackle big problems by automating tasks millions of times faster than a human can. Unfortunately, not all software is created equally. Much of it is whipped together without a plan, or is outdated, or both. Following the SDLC makes it easier for us to find issues before they are issues. If we stick to this process, we can use modern software engineering to change the world. Software engineering helps us analyze and design a global supply chain of food. Software engineering helps us connect emergency responders to their destinations. It helps us run airports safely and efficiently. Etc, etc.

<u>Thursday, June 28, 2018</u>

As I wrote my vision statement, I had to think about a problem that affects the world and how software could be used to fix it. I thought of a few options – a driving service similar to Uber where the driver drives you in your own car, a food donation service that allows you to ship raw ingredients to any country, etc. Eventually I settled upon addressing an issue that I see everyday – the prevalence of fake news and propaganda. A lie spreads faster than the truth does, and social media platforms like Facebook and Twitter are ripe places for fake news and doctored images to spread like wildfire before the truth even has the chance to catch up. The negative consequences of this misinformation are felt in elections and in governments around the world. Dictators like Vladimir Putin and Erdogan have mastered the craft of manipulating public opinion through social media. Bad actors use propaganda to justify eradication of opposition and assume absolute power. We know that dictators around the world do not lead to peace. And if fake news enables them to rise to power and maintain it, then fake news is bad for world peace.

This is a problem that I want to address by analyzing thousands of monitored accounts for trends and comparing them to new posts on Facebook and Twitter. Posts that have similar phrasing or keywords to known propaganda accounts, or link to similar websites, or originate from similar IP addresses will be immediately flagged as fake news when posted, rather than relying on users to report it as fake. Accounts that repeatedly post fake news despite being warned will have their accounts disabled or face financial penalties. It's not a perfect solution, but people need to know when they are being lied to and spreading them unwittingly.

Thursday, July 5, 2018

With July 4th being this week I haven't been able to write to you, oh dear journal. My sincerest apologies in making you wait. Today I had group meetings with both of my groups, one in which I am the customer and one in which I am on the software engineering side. I really enjoyed the meeting where I was the customer because that group came up with so many great questions and edge cases. As I wrote my software vision I liked my idea a lot but once I got 4 others pairs of eyes to look at it I realized how many details I hadn't thought about.

The first meeting alone taught me a lot about the process of software design and how implementation can quickly cause the scope of something to increase tremendously. However, after the meeting I liked my software idea even more, as now we had specific details mapped out. The idea I had was a fake news flagger on Facebook/Twitter that analyzes posts in real-time and flags them as fake news if they contain phrases, links, or images that have been found to be fake. Upon talking about the use cases and requirements with the group, the idea came together and was much more complete. Originally some of my ideas (such as notifying users about fake posts, banning repeat offenders, showing user reports, etc) were not specifically laid out but after the meeting we had a much better idea of exactly how things were supposed to work.

After the meeting where I was the customer, I had a meeting with my group where we met with the customer. The customer's idea was essentially a litter cleanup app that was essentially a hybrid of Pokemon Go (to get users outside and reward them with "points" for cleaning up), Endomondo (to challenge other users to clean up areas) and Waze (to rate users and verify that they actually cleaned up an area that they claimed to clean up). After going through the meeting where I was the customer, I felt much more prepared going into this meeting where I was on the software engineering side. I was able to come up with questions that nailed down the less specific details of the app so that we had a concrete understanding of exactly how things are supposed to work. Obviously, my group will still need to hash things out and we will have questions for the customer. In the group where I am the customer, I am sure new questions will arise. However this is all part of the process, and it's exciting seeing an idea go from a short summary to something with a real model.

Tomorrow and over the weekend I will write the use cases for my group project and get working on the message sequence chart. Both groups will be meeting on Saturday to go over what we've done so far. Jusqu'alors journal, je vous dis adieu.

Sunday, July 8, 2018

On Saturday my teammates and I worked individually on our contributions for the first homework assignment. I was tasked with defining use case 1, creating the message sequence diagram for use case 1, and creating the data flow diagram. In addition to this, I took the lead on formatting the document once everyone's parts had been submitted. As is usually the case with group projects, I took on a bit of a leadership role with doling out responsibilities, making sure that the requirements were met, etc.

In doing this assignment I learned a lot about just how big in scope a software idea is. Even an app or service that sounds simple has many parts to it that have to be defined explicitly. You can't just come up with a vague idea and implement features as they come to mind. There needs to be a clear vision first, with a clearly defined software solution based on that vision. The user experience needs to be defined before any coding begins. Otherwise, you will run into many issues, including incorrect implementation, missed deadlines, scope creep, missed features, and inefficient code. Other issues will arise as well. In other words – plan your shit out first. Treat your software like a 3-week trip overseas. If you wing it, you're f**ked.

In terms of the key questions of the week, I learned just how important it is for medium to large (or larger) software systems to have documented requirements. Because there are so many individual parts, trying to tackle the software problem without a plan will inevitably lead to the issues mentioned above. By not clearly defining all of the requirements and user interface/experience beforehand, you run the risk of greatly increasing scope and falling off track due to unforeseen issues. Late projects are never a good thing, so staying on schedule is important.

In addition, we need to make sure that the developers have a clear understanding of exactly what they are supposed to be doing. The person that designs the system from a bird's eye view will have a different approach to problem solving than the person that actually writes the individual components, so it is important that the designer is able to accurately define the exact requirements that the developer will be fulfilling. Requirements are researched and developed by talking about the software and breaking down the solution into smaller pieces. There will be a lot of questions for the customer to answer, because their understanding of how the system ought to work will inevitably be different from the software engineer's understanding. By clearly discussing all of the features in advance, we can address edge cases and other unexpected scenarios from the get go. Additionally, thinking about the various use cases and how data will be exchanged in the use cases helps us get a more complete picture as to exactly what will be required and what will not be.

If we are just adding a few features, we do not just need to add the requirements for the new features. We need to make sure that the new features will not get in the way of the existing architecture and that they won't break anything. With how many moving parts there are in a software system, it is very possible that new code can mess up what was already there. Because of this, we need to make sure that new features meet the requirements defined specifically for the new feature as well as the system as a whole. Enhancements to the product should be done carefully to avoid bigger issues down the line.

## Thursday, July 12, 2018

This evening we had our weekly meeting with the customer. Upon reviewing our work so far, the customer seemed to be pretty happy with our prototypes and requirements definitions. We had a few questions for the customer, which he was happy to answer and provide clarification on. There were a few things where our mockup didn't match exactly with the way that he had envisioned it, so we made sure to take note of the changes he wanted to see and split duties to update the prototypes and requirements accordingly.

I got to work on updating the use case and diagrams I worked on in HW1 and immediately realized how much work goes into changing requirements or specifications. Fortunately, we haven't begun coding yet. It was immediately clear to me how much worse a change to requirements or specifications would be after part of the system has already been built. This is why we hash over these details before development in tedious detail – to avoid having to backtrack after

I started thinking about the key questions for this week, and will only address the first one today. The advantages to developing a software architecture are many. First, you are able to map out all of the pieces for a software system before it is built, which allows you to create a design that will be both efficient and can support additional features and enhancements in the future. You are able to eliminate wasted resources and determine which functionality can be re-used throughout the system. Additionally, you make it easier to debug the system, as everything is broken down into logical components that have a defined function.

Essentially, designing a complete and robust software architecture adds time to the planning and design parts of the process, but it saves a lot of time for development and implementation, as well as debugging and maintenance. Since debugging and maintenance demand the most resources, it makes sense to create a system architecture first.

Sunday, July 15, 2018

Late last night our group received feedback on HW1 with several comments that mentioned why we lost points. Personally, I felt it was quite ridiculous to get this feedback with only 24 hours for us to fix before the next assignment was due. I had to spend half of my Sunday formatting a Google Doc that we used for collaboration into a document in ACM format on my computer. As someone with other obligations it was difficult for me to take the time to do this, and frankly didn't appreciate it. It was unfair. We should have received feedback earlier if it's something that will affect our grade for the following assignment.

Onto the key questions for the week. The first one was discussed on Thursday, but how about the ways that a software architecture can help manage a project? This was already touched a few days ago, but a software architecture helps manage a project by breaking it down into many components that each serve a purpose. One component could be some sort of functionality that can be re-used in various other components. There could be some page in the UI made up of 10 different components, each of which is broken down into simple requirements.

By breaking down the components into simple parts, we can spread the work around well so that different team members can work on implementation simultaneously and end up with a product where the code works together without any issues. If we expect individuals to contribute large parts of the system and design it on their own, there is a very high probability that something won't work with something that another user wrote. Having a system architecture that is clearly defined and logically split into components really helps with this. Also, as mentioned earlier, it saves a lot of time in the debugging and maintenance phases, as well as the amount of time it takes to create new features and functionalities.

Despite its usefulness, software architecture is often not used or ignored. This is for various reasons. One is impatience, big wigs often want the implementation that can be done soonest for the least money, not necessarily the one that will be done right. Sometimes project managers are not good at splitting up a problem into small pieces and attacking the issue that way. Some managers do not want to "waste" time in the planning phases and instead get straight to the implementation.

Essentially, I find that projects without a software architecture are those where the manager decided to do the easy thing now, even though it comes at the cost of spending far more time down the line. Many managers, especially on the business side of things, do not fancy themselves with how long things might take later. They only want to know how long or how hard something is right now. This mindset is particularly dangerous with software design, but unfortunately that doesn't mean that the mindset is uncommon.