Birth of a Program: A Condensed Journey Through the Software Lifecycle

An Honors Thesis (HONR 499)

by

Alexandra Curtis

Thesis Advisor

Dolores Zage

Ball State University Muncie, Indiana

April 2019

Expected Date of Graduation

May 2019

Abstract

Many software products are created to solve a problem, either for the creator or for a customer. Ball State University's Department of Computer Science exposes students to the latter during its Capstone course. In the fall semester, small groups of students seek out client partners with a business problem they can solve with software. The year is then a journey through the software development lifecycle from start to finish, culminating in a final working deliverable at the end of the year. I detail my journey through this formidable process, highlighting the milestones encountered and the lasting impact they will have during my career.

Acknowledgments

I would like to thank Marley, Jordan, and Michael for making this project possible, and for all of the unforgettable debates we had through the year.

I would also like to thank Professor Zage for showing us more about software development than we knew existed.

Process Analysis Statement

When one uses the product created by the Courier Management and Optimization Machine (CMOM) capstone team, they may not discover much about its path to existence. Navigating the web application won't uncover the months of design, or the team dynamics, or the painstaking testing process. The meetings and hours spent writing documents won't present themselves with a click of the "Login" button. Perhaps the most significant absence of all lies in the epiphanies learned and challenges faced, neither of which show up on the account dashboard.

The project can be traced back to August, when our team agreed to work with Diversified Mail Services (DMS) in Indianapolis, Indiana. Their company offers delivery and pickup services to local businesses in a similar fashion to their competitor, UPS. The small size of the business serves them well, and their steady client base cites stellar customer service as the main feature that keeps them coming back. We partnered with an analyst at the company who gave our team two problems. The first was that of their scheduling process. Company administrators were responsible for multiple spreadsheets containing pickup locations and courier driver routes. These employees manually created, updated, and maintained the sheets; a tedious process with too much upkeep. The second problem was that of how drivers validated their routes. Package pickup and delivery times were reported by the drivers themselves, which had recently led to the phenomenon of false reporting. A driver could claim that a route would take them 6 hours, for example, when it only took them 4. The remaining time would be spent taking extended breaks.

My first personal epiphany came during our first client meeting in September. It was made possible by having a few months of experience in my arsenal as a newly hired Software Engineer. This created a strange, limbo-like position in which I was being prepared to enter the workforce *after* I had been accepted into the workforce. It also created unique advantages, like being able to bridge the communication gap with our client.

Despite never having met, the client and I quickly developed a rapport throughout this meeting. He was up-front about what he needed from our product in the first few minutes, but a team member trudged through cookie-cutter assignment questions anyway. After being asked what color he wanted the website to be and what logo he wanted, I sensed frustration in his voice. He had little to no interest in visual design; he wanted to discuss business problems. Though we needed the answers for our interview documentation, it was painfully clear that the questions were not leading us in a helpful direction.

I waited until the pre-made interview questions were exhausted before attempting my approach. My initial strategy was elaboration, repeating his statements in my own words to establish shared understanding (and prevent miscommunication). It also showed him that he was being listened to attentively and his words were understood. The positive change in his demeanor was almost tangible, so I pressed on.

After establishing the higher-level objectives (i.e. the program's desired function in plain words), I started to dig toward the more technical items). For each feature

discussed, I stuck with the process of parroting his responses in my words to clarify his intention. At this stage I quickly realized that another skill was coming into play.

When a software company partners with a customer and program requirements are requested, a divergence occurs. Business partners are not expected to have proficiency in technical jargon, just like all software developers are not expected to understand business terminology. This dynamic is the basis of an entire career in the software industry: the customer liaison. While this person may not be heavily experienced in software creation nor upper-level business management, their education covering both sides of the divide helps them translate needs and bridge that gap.

After taking the helm in our meeting, I became aware of this role's vital importance. The client was focused on outcomes, not small details of execution, and this required that I challenge myself. When I thought of a question, I would quickly alter it before asking. Instead of asking what features our application should have, I asked what he wanted to replace in his current system. To find out what fields we needed for our database, I asked what information they requested from clients.

My line of questioning eventually exposed a drawback. Some objectives had multiple possibilities for development, and my questions helped me illuminate a path. In an ideal situation, all team members would be involved in a discussion to explore possibilities (especially ones I hadn't thought of) before requesting details. Instead, I had to make snap decisions on the details to gather so we could keep our meetings short, out of respect for our client's time. This could have been remedied if we had more access to

the client, allowing more frequent and/or longer meetings, but the company was in Indianapolis and the client had a busy schedule.

After establishing client requirements, our team held two meetings. The first was a general, non-technical meeting to confirm a collective understanding of the client's desires. During this meeting I would explain the understanding I had, and then we would come to an agreement on what the client's intentions were. The second meeting marked the point in which we dug our heels and debated how to accomplish these objectives. Technical details weren't yet discussed, but we would ruminate upon program structure and approach.

I took a backseat during these meetings save moments where we hit proverbial walls, or I elaborated on a teammate's idea. When first dealing with the client I saw a moment where someone had to step up, and I made the executive decision to do so. Because there had not been a previous discussion appointing any single person as a liaison, I was wary of my actions being taken as domineering. Only recently in my life had I learned to step back, collaborate, and listen – and it was the result of hard work over many years.

Indeed, like many self-driven students in group projects, I used to entertain the notion that I knew the "best way" to do a project. At my most ignorant, I would simply *do* the project. These incidents occurred in a primary school class of less than 40 students. I had the blessing of being well-liked by most of my peers, so it was accepted by others as an unburdening of responsibility. This propagated the problem for a long time.

Then, in the 7th grade, I put great conscious effort into changing how I approached group projects. Our history class split into two groups with the task of creating and performing a skit. I was quickly looked to for ideas. Despite having many fully detailed plans in my head, I held back. I offered a single idea - the setting of a game show - and then switched focus to encouraging and embracing the ideas of others.

To me, that one small project will forever stand out as the day I became a true leader. Not a good one, mind you, but I realized there were new skills to hone - mediation, conflict resolution, how to have a construction debate (and reach tangible outcomes), as well as mutually beneficial task delegation. Years later I would attend leadership conferences on behalf of my school, captain both the JV and Varsity volleyball teams, and win several mental attitude awards. I did not start out as a good leader per se, but I put a lot of work into improving that facet of myself. Despite all of that effort, I still do not voluntarily take leadership positions, and prefer to work alone if given the chance.

In fact, more often than not, I find that I "end up" in leading roles rather than asserting myself within them. Based on my anecdotal experience, self-appointed leaders in student groups specifically are less likely to foster successful teams. An air of self-importance poses a risk, and often leads to a lack of respect. This point brings us back to the project at hand. My actions in our first client meeting could easily have been interpreted as a forced-leader move, especially without prior consultation, and I wanted to avoid establishing this group dynamic at all costs.

At this point we knew the details of our project but were tasked with many academic assignments regarding project management and software design. I heard

innumerable gripes from all students about the extensive planning and lack of producing. Initially, I contributed gripes of my own. My discontent was incited by the unusual position I was in. Class was for preparing me for the industry while I was already *in* the industry. However, as I found later, it was also this unusual position that quelled my discontent.

In the software development lifecycle, there are several options regarding process. Many different methodologies exist, such as waterfall and agile, as well as different project management styles like sprints and Kanban (Hoffer, George, & Valacich, 2006). Design documents can be done in UML or use case flowcharts. Sometimes, due to pressing deadlines from the client, design documents are bypassed completely. In true practice, many of these approaches are "Frankenstein'd" together. After experiencing the reality of large-scale software product creation, I resented my academic experience. The "perfect" learning environment had an unrealistic (and seemingly arbitrary) process. It was a common whine of students: "Why are we learning this?" However, after learning more of my organization's structure and the roles within it, I changed my mind.

It should be stated that no company does everything perfectly. In the software industry, many large and well-known companies are looked up to for their execution of certain aspects of their organization. In my work, I've been directed to learning resources that cover certain practices of Spotify and Google engineers. Their team structures were executed well and we wanted to learn from that. The same thought process exists behind learning the "ideal" way to do things in a school setting. Real-world experiences may not be "by-the-book" at every turn, but knowing the book is a great place to start.

Anecdotes of previous capstone students led to another great learning moment. They described working on their projects from the first month of classes, and said the majority of teams reached completion. I envied their freedom to focus on their development without additional requirements and assignments. At one point, it hit me that I was being selfish. A degree in Computer Science lends itself to a staggering number of jobs. If the search term "Computer Science Degree" is entered into Indeed.com, it returns 181,072 positions at the time of this writing ("Computer Science Degree jobs," 2019). Clearly there are many paths one could follow upon finishing their education.

Despite my initial envy, I felt these teams had gotten short-changed. Sure, I didn't enjoy the assignments about team management scheduling and project workflows, but those topics aren't my domain. I didn't expect them to be included in my future responsibilities. With that said, how can I assume learning them is a "waste of time?" Someone in that class could move on to a position in which these items are dependent upon their expertise.

Those sentiments bring me to the significance of this project. The journey to its conception was not that of a simple program to be cranked out in a few weeks. It was an undertaking that paralleled a real-world timeline - though compressed into a far shorter duration. Because so much had to be done in this time, teams of three to four people tackled objectives that could be assigned to entire departments. While some of the lessons learned likely weren't relevant to everyone, they were necessary to create such a consolidated immersive learning experience.

Naturally, such exposure does not come without its challenges. Design-specific assignments early in the course (such as sequence diagrams and use case tables) were completed by our team out of necessity - not for viability. Team members had pressing responsibilities and conflicts, life priorities, and other difficult courses on their plates. We knew what our overall program would consist of early on, but we hadn't deeply discussed how to get there or with what tools. The return from winter break and the official shift of teams to the programming process led immediately to the splintering of our team. It was an event we overcame but never truly recovered from.

One member of our team took it upon themselves to put the entire foundation of the program in place without sharing their plans. The rest of the team were given instructions to download a specific development environment, numerous plug-ins and libraries for the code, multiple separate programs to run the backend database, and then were told to submit changes through Git version control. Getting the rest of the team adequately set up took an inconvenient amount of time, and it only worsened from there. Despite our appreciation of the effort and the progress made, it was clear that our group dynamic had shifted. We now felt far removed from the accepting and collaborative atmosphere we'd had before - one person had unanimously established themselves as the leader.

When it became clear that this incident ultimately hindered the productivity of the group, it was raised. The extensive code in place had no comments, and it was not written in an easily understandable way. One of the plug-ins more heavily used was based in a language entirely unfamiliar to two members of the team. Because of numerous

dependencies already created, nearly every change or issue had to be run through this one individual - the antithesis of parallel team development. These points were brought to light in a non-confrontational and honest way with the desired outcome of equal collaboration. This discussion was not received open-mindedly and, in fact, resulted in hostility. The rift had formed.

As time went on, respect dwindled. The team kept level heads and mature dispositions. Communications stayed cordial…for a time. Increasing passive aggressive behavior and one confrontation later, the rift turned into a fracture. Group communication continued but a separate channel was created for the other three members. Mounting stress from responsibilities both inside and outside the classroom led me to disengage from many programming aspects of the project. My patience was dwindling, and I found myself biting my tongue more often. However, I learned that my sense of commitment ran deeply. I could never turn my back on my team, and I began to find ways in which I could still foster our success.

Looking back, I should have shelved my personal feelings and made more of an effort to reconcile the group. Mending the rift may have been a monumental task but I feel the outcome would have been worth it. Instead I took to self-preservation, as I felt my mental health was better protected by focusing on what I could control. In the team dynamic I took on the role of assignment responsibility and client communication. When class work was assigned, I often completed a large amount and handled submission to enable continued development by the team. I found that reading numerous technical documents during my employment gave me a keen eye for writing them, and I showcased

that strength as best I could. When it came to client communication, I felt I was shouldering one responsibility the team wouldn't have to fret about. In fact, during one meeting with our client, an incident occurred that usually hampers product development. It has a history of setting teams back and sometimes forces groups back to the drawing board - changing requirements from the client.

In part due to circumstances previously described, we knew there would not be enough time remaining in the spring semester to finish features we'd discussed with the client. Prior to expressing our concerns, however, the client informed us that his priorities had changed. He was considering other options for the problem domain of driver confirmation, so our team would benefit him most by focusing on the database system. It was then agreed upon that we would deliver those features in a well-tested and polished state rather than add potentially unstable new features. This changed our subsequent development schedule and instilled confidence amongst our team that we would deliver an acceptable product.

The preparatory work behind the given examples of this project includes the creation of a web-based API (Application Programming Interface) that routes commands to a MySQL database. In this form, a user sends a command to the API by using visual components on the website (like the login button, for example). The necessary information is then passed through to the database in a CRUD-like system where the user can create, read, update, or delete items ("What is CRUD?"). Our code also defines methods for such user actions (i.e. the 'joining link' between CRUD actions and the

database) as well as so-called “blueprints” for each type of user (like administrator or courier) and the components that are visually shown on the web page.

The events my team and I experienced during this project were significant to our professional and personal development. I was fortunate enough to receive lessons and personal insights that I would not have been taught if working alone. How I handled those moments and what I learned from my actions are memories I will take into my professional career, where I will build upon them and continue to learn from new situations.

Bibliography

"Computer Science Degree jobs." *Indeed*, April 2019. Retrieved from: https://www.indeed.com/q-Computer-Science-Degree-jobs.html

Hoffer, George, & Vlacich (2006). *Modern Systems Analysis and Design*. Pearson Prentice Hall.

"What is CRUD?" *Codeacademy*, N.D. Retrieved from: https://www.codecademy.com/articles/what-is-crud