Software Development Journal

**Initial Premise & Motivation**

*For the Application*

I have always been a hands-on learner. For that reason, I have always found it kind of unfortunate that the main method with which information is conveyed in college is via lecture. Now that I am nearing the end of my undergraduate studies, I can truthfully say that, to me, a useful lecture was a rare occurrence. To be frank, I can count the number of classes where lecture was useful on one hand.

For that reason, I need to go out of my way to study to understand and retain any course-related information at all. I have researched several schools of thought on the concept of self-study, and have tried many methods of doing so. The strategy that I have had the most success with goes as follows:

1. Obtain a piece of course material to study, usually a slideshow used in lecture.
2. Ask myself questions about the course material as I am reading over it.
3. Write down these questions and the answers to them.
4. Use the questions and answers obtained in (3) to make a quiz.
5. Quiz myself until I get all the answers right.
6. To better retain and understand content, return to (5) at a later date for review.

In order to expedite this process, I looked at some services and software that allow the user to quiz themselves. The best one I could find ended up being Quizlet, which is a web service that allows the user to create a set of terms and definitions to quiz themselves with

(think virtual flashcards). Ideally, I wanted a lightweight, open source, local application that allows the user to quickly create quizzes about any type of content with a simple, intuitive GUI and/or syntax. I was unsuccessful in finding anything that met all of these criteria, so I thought to myself, “Why don’t I make it?”

*For this Journal*

Whenever I have a problem that I need to solve via programming, my standard process is as follows:

1. Identify and write down the problem to solve.
2. Check if the problem has already been solved; if so, reuse the code that solves it.
3. Check if the problem can be solved using a function; if so, write the function to do so.
4. Check if the problem can be solved using a class; if so, split the problem into subproblems to be solved using members of the class.
5. Check if the problem can be solved using a collection of classes; if so, split the problem into subproblems to be solved using classes.

The actual writing of code only takes place in problem (3). When I solve this problem, I use the first solution that comes to mind. Oftentimes, this solution is not the best, simplest, or easiest solution. This solution may even have no redeeming qualities other than that it solves the problem. There may be a great deal of better solutions to same problem. Maybe the solution doesn’t even actually work.

All of these are obviously negative possibilities, but that’s not the reason I use the first solution I think of. I use this solution to better understand the problem I’m trying to solve in the first place. Although it is not necessarily immediate, after writing this first solution, I almost always think of a better one. Sometimes, I don’t think of a better solution for a long time (weeks, if not months). And when I implement this better solution, I end up with less code than I started with. Often, I’ll implement another, even better solution after that, and end up with even less code.

You’ve probably figured out where I’m going with this by now. The more I work on my project, the less I have to show for it. So, what’s the solution to this?

1. Keep the initial code for every problem you solve because it does the job (even if it does it poorly), resulting in a larger, more impressive-looking codebase.
2. Continue replacing solutions with better ones, and accept the fact that the codebase will grow slowly.
3. Create some other deliverable to document the mental labor, thought process, and learning behind the project itself.

The first solution I tried was (1). Unsurprisingly, my experience went poorly. Indeed, I was able to implement the first few software functionalities surprisingly quickly. The codebase grew at a rapid pace. However, implementing each functionality became more difficult than the last, since I took no time for maintenance. Bugs were rampant, and finding their source was incredibly frustrating. Determining an ideal way for functionalities to interact was practically impossible when I did not fully understand what the functionalities were doing in the first place. Code was repeated in multiple places in multiple files, so changes to one instance of repeated code were not reflected in the others. In short, it was a mess.

For that reason, (1) was a completely unsustainable approach. I switched back to (2), which is my preferred approach, and worked on the project with great success. However, I couldn’t shake the feeling that I would go to defend my thesis, someone would see a few files and a few hundred lines of code and go, “That’s it?” I expressed my thoughts to my thesis director, Professor Miller, and he proposed that I create this very journal. That is how we arrived at (3), which I think to be the best approach for this project, even if it results in less time spent programming.

**Lost Entries**

Technically, no entries were “lost” in the sense that I created them, and then lost access to the file containing them at some later point. However, during the first semester of this project’s development, I worked on the project with no version control system, and did not produce any external documentation. For that reason, within the scope of the project’s development, there existed potential for me to create much more entries than you see here in this journal. This is truly unfortunate, because in this first semester, I was in a much worse state of mind than I am now – which would have made for great journal material.

**Retroactive Entries**

*Introduction*

This journal was created quite late in the project’s development (second to last month). For that reason, the majority of the work done on the project has already been completed. Obviously, I cannot directly capture my state of mind as it were earlier in development. However, between the first and second semester of development, I (finally) taught myself how to use GitHub, and create a repository for this project for the purpose of version control. Every update of my code has a short description to go with it. These entries contain the short descriptions themselves, their dates, and some comments from present me to give context, explanation, or my thoughts.

**Journal Entries**

*March 2nd, 2021*

Today, I created the software development journal itself, so this will be its first genuine entry. I created the file, came up with a general structure for the document, and started typing away. I wrote the journal’s first section, “Initial Premise and Motivation,” which is currently comprised of two subsections: “For the Application” and “For this Journal.” I think they’re both quite good, and should serve as a great introduction to the journal – let’s hope I think the same in a month from now!

I also wrote the section “Lost Entries,” which discusses the work that was done on the project pre-GitHub. After that, I wrote the introduction to the section “Retroactive Entries.” I hope to finish this section tomorrow and capture my thinking from my previous commits. Of course, commits made from this point onward will be captured in the entries in this section here.

On that note, today’s thesis work consisted of a few hours of thinking and writing. The only thing I will commit today is this journal itself, although I’m not sure how GitHub handles docx files. My plans as of now are to finish up this entry, give this document a quick proofreading, and then commit this document to the project’s repository. Since all of today’s work took place in the journal itself, this entry is definitely an example of meta-journaling.