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 does not even actually work.

All of these are obviously negative possibilities, but that is not the reason I use the first solution I think of. I use this solution to better understand the problem I am 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 do not 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 will implement another, even better solution after that, and end up with even less code.

One can probably tell where I am going with this by now. The more I work on my project, the less I have to show for it. So, what are some solutions to this?

1. Keep the initial code for every problem solved 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 could not 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.

**Minimal Requirements**

At the time of submitting the prospectus for this project, I did not fully understand the workload that awaited me in the following year. For that reason, the initial requirements and timeline outlined in the prospectus were unrealistic. A more realistic, minimal set of requirements for the software will be listed here. This journal also exists as a way of supplementing the work done for this thesis.

1. The user shall be able to write and design a quiz in a plaintext (.txt) format.
   1. The syntax for writing this quiz shall be simple enough such that it is accessible for non-programming people.
   2. The language with which quizzes can be written shall allow for the user to write questions, answers, and feedback.
      1. For a given question, the feedback section shall be optional.
   3. The language with which quizzes can be written shall allow for the user to provide randomly selected variables to be used when presenting the quiz.
      1. A randomly selected variable shall be given a name and a list of random values to choose from.
2. The software shall be able to scan a preset directory for quiz files, and present these files to the user.
   1. The software shall ignore any files in this directory that are not labeled with the extension “.txt.”
   2. The software shall label text files found in the quiz directory with an integer, and present this labeling to the user.
   3. The software shall prompt the user to select a quiz by entering the integer used to label the quiz.
   4. The software shall return to the quiz selection screen upon completion of a quiz.
   5. The software shall return to the quiz selection screen if there is an error while trying to parse the quiz.
   6. The software shall exit if the user enters an integer that is not a valid label for a quiz.
3. The software shall be able to parse a quiz file, and present the quiz to the user.
   1. The software shall be able to parse a quiz file, and store the data of the quiz in memory.
   2. The software shall present the user with descriptive error messages if it encounters an error while parsing the quiz.
   3. The software shall be able to sequentially present the user with questions of the quiz.
   4. The software shall be able to sequentially accept the user’s given answer for each question.
   5. The software shall be able to check a given answer against the correct answer.
   6. The software shall be able to provide the user with feedback regarding whether or not the user’s given answer is correct or not.
      1. This feedback shall be able to be presented in two ways: immediate, and post-quiz.
   7. The software shall be able to provide the user with optional feedback written by the creator of the quiz if the user’s given answer is incorrect.

**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 can be seen here in this journal. This is truly unfortunate, because in this first semester, I was in a much poorer 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 created 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 are both quite good, and should serve as a great introduction to the journal – let us 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 am 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.

*April 6th, 2021*

As one can tell from the difference in entry dates, I certainly did not “finish this section tomorrow” as was planned. This is because shortly after the last entry, my schedule was bombarded with midterms, assignments, and projects, to a magnitude that was somewhat… unexpected. But, it has all been said before; I am here now, so let’s move on, and get down to brass tacks.

At the time of writing this, the project is incomplete. It is in a workable state, but it is definitely not complete. However, it is close – with a little bit more work, the project will be in a state that I consider to be satisfactory. Although, this feeling of closeness may fall under the category of “so close, yet so far.” To quantify exactly what I want to do for the project, a section will be added entitled “Minimal Requirements.” I have discussed what can be accepted as a set of minimal requirements with my thesis director, and have taken some rough notes on the subject. I would like to write this section tonight.

As previously stated, this software development journal was initially made as a way of compensating for the fact that programming, and following good practices while doing so, often results in a small codebase. However, what has not been said as of yet is that, at this point, the focal point of this project has shifted away from the software itself, and towards this journal. The thesis has gone from a software development project, to a “learning about software development” project.   
 What does this shift-in-focus imply? Does it imply that the thesis is a failure, since development of the software did not go as far as planned? Absolutely not. A set of revised minimal requirements were defined for the software. At this time, the software is on track to meet these requirements before the defense. Even then, if by some stroke of bad luck, the thesis was not accepted outright at the time of its defense, there will still be ample time for revisions.

Another question comes to mind: suppose that software development went as planned. How much of a difference would this have made? I must admit that yes, the software would have turned out significantly better than it is now. Looking back at the prospectus summary, the requirements were somewhat unrealistic. Would it be possible for an experienced programmer to create such an application in a year’s time? Definitely. Was it possible for me to create such an application given a full course load, a capstone project, an accelerated graduate program, and additional honors work? It was not.

To summarize: although the thesis will not be up-to-par based on the original standards, it will be complete by a set of more reasonable, redefined standards that have been clearly outlined in a different section. Additionally, the focal point of the thesis has shifted to this software development journal, which was something that was not even defined in the original prospectus. At this time, I am going to break away from this entry, and write the minimal requirements section. After this, I will return to this entry once again, log any additional thoughts, and then decide where to go from there.

I have now completed the sectio

-write section

- return to here

- mention that you took a break

- mention that you read over entire document