Spring 3-12-2018

An Analysis of Project Setup and Organization in Software

Adam Fitzgibbon
University of Nebraska-Lincoln

AN ANALYSIS OF PROJECT SETUP AND ORGANIZATION IN SOFTWARE

An Undergraduate Honors Thesis
Submitted in Partial fulfillments of
University Honors Program Requirements
University of Nebraska-Lincoln

by
Adam Fitzgibbon, BS
Computer Engineering
College of Engineering

March 12, 2017

Faculty Mentors:
David Keck, PhD, Engineering

Table of Contents

Abstract

The setup of a software project can greatly influence how efficiently software can be built. In this paper, I will be discussing my experiences with various software projects and their setups. These experiences will be used as a basis to draw conclusions on the strengths and weaknesses of specific project setups in certain situations.

Because of the complexity of building software, there isn't a project setup that works optimally for solving every type of software-based problem. The best approach is to know what variations would best fit the situation and make design decisions from there.

Introduction

By its nature, software is complex, but even more so to build. Thus it is important for those creating it to not only closely consider their project structure (architecture), but also the style and environment before even starting the development. Collectively all of these aspects of a software project will be referred to in this paper as the “project setup.” Through my experience on software projects during my time pursuing an undergraduate degree at the University of Nebraska-Lincoln, I have had the opportunity to observe various project setups. This paper will discuss the strengths and weaknesses of these project setups by analyzing how effective they were in my experience and by looking at other people’s experiences with similar setups.

As the purpose of this thesis is to reflect on my experiences in Raikes Design Studio (the Raikes School version of senior design where we work on a team to develop software solutions for company sponsors), the main focus will be comparing and contrasting the two very different projects I had over my two years of Design Studio. However, for additional perspective I will be using experience from projects I have worked on as part of summer internships as well as my research position at the Holland Computing Center here on campus in the Schorr Center.

Experiences

My first year in Design Studio I worked as a developer on a project for TDAmeritrade. The goal of this project was to take common process within their system and automate it. Specifically, a lot of the steps for handling internal transfers between TD accounts required a phone call and an employee going down a spreadsheet to make sure there were no restrictions on the account. Over the course of the year we built a rules engine that automatically made the account restrictions checks. The idea was that a TD software team would be able to take our backend code and access it from their web frontend to allow their customers to make internal transfers at their convenience from their website.

For the second year, I was the development manager on a project for Fiserv. This project focused more on providing a working prototype rather than code that is meant to be directly integrated into the sponsoring company's system. The goal for this project was to solve the problem of bank managers having to go through subordinates to view data on their customers. To address this we were tasked with using voice recognition software so that managers could easily bring up reports by simply using natural language rather than having to be familiar with a query language or by going through someone that does.

To accomplish the goals of the project, we had to integrate two of Microsoft's software products into one application. Cortana was used to handle the natural language processing and PowerBI was used to generate and publish reports based on obfuscated banking data. The idea was to provide a proof of concept that Fiserv developers could use as a reference for later implementing themselves within their codebase.

While the focus of this thesis is to reflect on my Design Studio experience, I will also reference my other relevant experience in order to create a larger sample size of projects to discuss. I have worked two separate summer internships that can contribute here. My first internship at the local e-commerce company, Spreetail, had me working on their internal software. This software was basically a large .NET web app that all of the other departments in the company used to do their jobs. My second internship was with the GPS and wearable

company, Garmin. Here I got the opportunity of working on the Android companion app for one of their watches. This gave me a unique perspective as I had never worked on mobile development, and certainly not for such a large product.

Finally, my experience as a developer at the Holland Computing Center within the Schorr Center has gotten me involved in a multitude of projects that would be useful to bring up. These range from simple scripts to control decorative LEDs all the way to collaborating on a Angular web app to provide a graphical interface to interact with the computing clusters with.

Discussion

After taking into account my past experience, I can start drawing conclusions about how the setups of the various projects affected the development. When looking specifically at my time on the TD team, we were working within their actual codebase. The benefit of this was that it allowed for them to keep their proprietary account data within their system so that there were no security concerns. As a side-effect it made the code handoff excessively easy as all we had to do was brief them on how the software worked as they already had access to the branch that we were pushing code to. Additionally, coding style and architecture were already enforced so we just followed their guidelines during actual development. In retrospect this was actually an effective project organization as it gave us a lot of freedom to solve the problem while still providing design constraints which helped to guide us in setting up the project in the beginning.

The setup of the Fiserv project was in many ways the opposite to TD's. Instead of working within an existing codebase we created our project from scratch. As the development manager for the project, this proved to be a big challenge as I was tasked with looking at the project description and figuring out what would be the best project setup to start with. We actually had to move away from the architecture I initially settled on due to technological limitations we discovered later into the project. Because we were working with bleeding edge software that was still being updated by Microsoft throughout the year, we had to be very flexible in adapting our project setup to accommodate the technology we were working with. I would say this was the main benefit of working in a small new environment as opposed to the large enterprise codebase of TD. The ability to stay flexible and being able to quickly pivot as we saw fit was invaluable to the exploratory nature of the project. The downside of this method is that constantly having to change your project structure comes with a lot of overhead that could possibly be avoided if we would have been able to just stick with one setup from the beginning. Our developers ended up having to recode functionality that they had already programmed because of changes in technologies that were available to us. It is unlikely that

these inefficiencies could have been avoided without having in-depth prior knowledge of the very new technologies involved, but it is something to consider when looking at this style of project in general.

One thing that is constant across Design Studio is employing agile development practices in order to have a standard process no matter the project. This is a widely used and often effective methodology that promotes swift development. It's particularly effective in that it is possible to scale these development practices to any size of project if executed correctly. (Comella-Dorda, 2018) So I think the majority of the time, an agile approach is going to be the right one regardless of the other factors of the project setup.

Under my analysis, my two Design Studio experiences can be thought of as opposite sides of the spectrum. They aren't extremes by any means, but they do represent two very different styles of software projects. A preponderance of my other project experiences can be placed somewhere on this spectrum. As a rule of thumb, most industry projects are going to be closer to the TD experience. You are going to be working within the company's codebase, but the style and structure will be different depending on the company. Based on my experience with Spreetail, for instance, they would fall on the TD side of the spectrum but with less stringent rules on coding style. While they still enforce architecture, there are not as many checks in place. This take makes sense for a relatively smaller company that isn't quite at the benchmark of what I would call an "enterprise level codebase". This style of project structure works for them because in the early days of a company being able to quickly push out essential code is crucial to its success.

On the other hand, the mobile project I was working on at Garmin was perhaps more rigid in their architectural and style requirements than TD. Every feature was precisely mapped out by a member of the user experience team, and all code had to be reviewed carefully by multiple people. Architecture and style was very strictly enforced. While this sometimes made it more difficult to develop features quickly, the caution was warranted as it was a customer

facing project. Any mistakes could create a bad user experience which might lead to a loss of customers or compromise of the customer's data.

While industry software has to meet a certain expectation of software quality, my experience was that in research that wasn't necessarily the case. That is not to say that it was encouraged to disregard architecture or project organization, but often times the overhead of thinking about those things wasn't worth it when a lot of the time a medium sized script would do. For example, my first project was working on a script that controlled LEDs to create a fun display for the cluster room. This project really didn't require a full software solution, but rather a couple of Python scripts. This style of project worked in this instance because in general they would only have one developer working with the code at a time so separating out functionality to enhance collaboration was not a priority. While controlling LEDs might seem like a mundane example, this style of project showed its benefits when I was working on performing RNA sequence analysis on rice and wheat using the statistical analysis language, R. Performing this analysis was a very linear, but computationally intensive process, so the best way to do it was to write the analysis code in a single R script and then run the code over large amounts of data via the computing clusters. Again, a more complex project setup would be unnecessary here and just make the process of running the code on the cluster more difficult.

Architecture is also an important factor to take into account when considering the best way to setup a software project. There are a lot of common patterns that are used, and the article "10 Common Software Architectural Patterns in a nutshell" does a great job of giving a description and diagram of many of them. (Mallawaarachchi 2017) The patterns I am most familiar with are layered and model-view-controller. Often times they are very effective in multiple situations, but it's still good to be aware of the other common patterns in case a situation comes up where they might be more effective.

Conclusion

All of the things that must be considered when thinking about software makes it complex, so it follows that there is not always a straightforward answer to software-related problems. This is the case when looking at the best setup for a software project. Through my experiences in Design Studio as well as industry and research, I have found that the best way to approach a project is to analyze the end goal and what is required to get there. Of course it's possible to apply the same setup to almost any project, but in the end you will be fighting as much against the project itself as you are against the actual development.

In the case of larger projects where multiple developers need to collaborate, the best course of action is to front-load the effort and establish a consistent architecture from the start. This is particularly effective when the technologies being used are well-established, making it easy to predict any potential roadblocks in the future. For these, having that existing codebase already provided is a large boon, as it means that all of those technical decisions have already been made. Of course on the other end, smaller projects with less collaboration are going to naturally lend themselves to be lighter in project setup, allowing them to be much more dynamic in build.

It is really up to the technical lead of a project to make the right setup decisions. Doing so can make the difference between a successful project that meets all of its goals, and one that trudges along underpromising and missing deadlines.

References

Comella-Dorda, Santiago, et al. “An operating model for company-wide agile development.” *McKinsey & Company*, www.mckinsey.com/business-functions/digital-mckinsey/our-insights/an-operating-model-for-company-wide-agile-development.

Mallawaarachchi, Vijini. “10 Common Software Architectural Patterns in a nutshell.” *Towards Data Science*, Towards Data Science, 4 Sept. 2017, towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013.