Lessons Learned: Building Software for the Waterloo Engineering Endowment Foundation[1]

Andy Toulis

December 8, 2015

[1] Term paper for MSCI 342: Principles of Software Engineering taught by Dr. Mark Smucker.

Introduction

This paper outlines the lessons I learned and contributions I made over the term as part of a software team revamping the Waterloo Engineering Endowment Foundation (WEEF) website. WEEF donations are used to fund "undergraduate laboratory equipment, student projects, computer upgrades and academic tools/teaching facilities."[2] My team employed a Scrum methodology to improve the WEEF process which is controlled by a rotating student director and the administrator Mary Bland who was the primary stakeholder for our work. Figure 1 below is the full sprint plan that illustrates all improvements aimed for by my team. My contributions are highlighted in this paper, including implementing search on the website[3] and allowing Mary to manipulate allocation records online.

[2] Waterloo Engineering Endowment Foundation. About weef. URL http://www.weef.uwaterloo.ca/

[3] A stand-alone website was built and will be adopted by the Winter 2016 WEEF director.

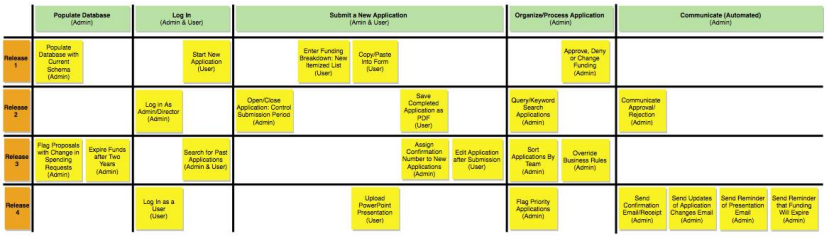

Figure 1: Breakdown of sprints one through four.

Learning and Contributions

To reflect on what I learned over the term and to discuss my contributions, I will focus on three main pillars of MSCI 342: working as a software team efficiently, designing software to solve customer problems and building quality software.

Working as a Software Team Efficiently

To improve the overall efficiency of my team, I self-assigned to jobs that everyone depended on to do work, including setting up the project database in `MySQL`[4], populating it with records and creating a connection function in `PHP` that all code would point to.[5] I have extensive experience using databases and therefore took lead to set

[4] `MySql` was selected for a database since the Management Sciences department had a server available. Furthermore, it is a free environment to operate on.

[5] Some teammates decided to simply copy the function instead of taking my advice to use "include" in `PHP`.

my team up. I also was able to deliver same-day results when we needed to migrate databases due to issues with up-time.

An additional step I took to make our team perform better was sending out an anonymous Google Documents form for them to complete prior to our sprint retrospective meetings. The form had three simple questions which focused on things to keep doing, things to stop doing and things to start doing.[6]

This term was the first time I have used a shared repository with a team. The service we used was GitHub (Git), as we were provided with a private repository.[7] This proved to be a major improvement over previous practice I have had in other group settings. Efficiency was improved since cloning and updating the build was managed by Git. Furthermore, the time wasted from having code conflicts was eliminated since conflicts between code on my team needed to be resolved before pushing to the build.

I recommended my team to use branches on Git to avoid breaking the build, but due to time limitations in learning new tools, my team did not adopt this best practice. I still decided to use branches to ensure I never broke the build, which was a challenge faced on several occasions.[8] By using a shared repository to effectively manage code, this project served as a learning experience for how my future code bases should be managed.

[6] My team was required to mention one thing to keep doing, which is a trick for maintaining team spirit that I learned through MSCI 342.

[7] Formal communications were made on a private Piazza group.

[8] The build broke several times before stakeholder meetings partially due to having incomplete features on the Master branch.

Designing Software to Solve Customer Problems

Solving problems is at the core of software engineering. I solved one of the biggest problems for WEEF which was implementing a fast, adaptive and easy to use search bar. Not having search was one of the biggest bottleneck for all members using the WEEF site, students and professors included. To tackle this problem, I first broke it into smaller pieces, a skill I learned in my Algorithms & Data Structures course.[9] The first step I took was to create a form to obtain a user's query, which can be seen at the top left of Figure 2 below. User inputs from this form would then be used to query the database.

[9] Algorithms & Data Structures was the previous software-based course I took, also taught by Dr. Mark Smucker.

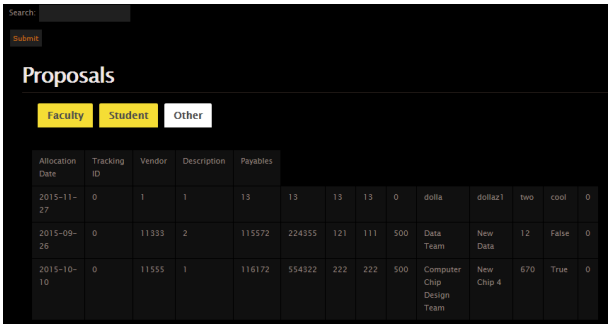

Figure 2: The landing page for viewing proposals to WEEF with a search bar.

Once a user submits a query using the form, multiple steps must occur before the query can be used to extract records from the database. Firstly, the query string is cleaned and stop words[10] are removed from it using a custom-built function. Next, since multiple columns[11] can be of interest to a user and since there are multiple words in a given search query, simply writing a `MySQL` statement to query the database is not possible. Instead, I wrote a script to compose a `MySQL` query logically based on the conditions given by a user.

The final search implemented performed very fast (in less than a second) and was adaptable to dirty, real world inputs. It scaled well as the website became more complex, being used across multiple parts of the site, including the editing page for Mary. In the future, an improvement to the approach would be to take into account information retrieval features such as the frequency and inverse document frequency of each term to rank the relevance of the results. The result of querying "Data" is shown in Figure 3 below.

[10] Stop words are words like "a", "the" and "I" which do not add specific value to a search query.

[11] Currently, only the vendor column is being queried but it is predicted that other columns would also be important, such as team name. Therefore a generalized querying procedure was written to allow for customization in the future. This aligns with the notion that users do not always know what they need.

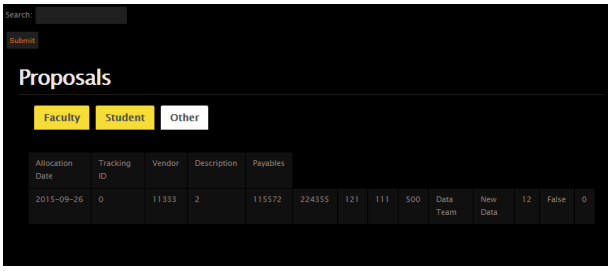

Figure 3: The result of querying "Data" on the proposal viewing page.

Another major problem I solved was allowing Mary to edit allocation records online. I built form and handler code for editing. I also built the original code which the handler pointed to for performing record manipulation in the database. This was later replaced by similar structured work that was in our final product. The handling page for inputting changes can be seen in Figure 4 below.

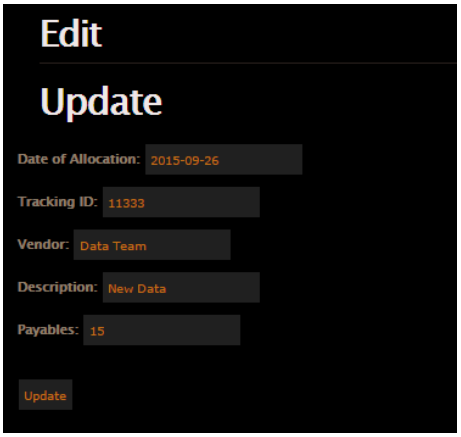

Figure 4: Editing an allocation record.

Building Quality Software

While replacing the database, I noticed our pages were not error-handling correctly when my connection code was not implemented and when there were no records in the new database. In response to this, I sent a reminder to my team that we need to adhere to the best practice taught in MSCI 342 of owning our own quality as developers. In this discussion, I pointed my team to code that contained standard `PHP-SQL` error-handling, such as checking if a query returned any results.

Finally, I posted an open note to my team about how passwords should be stored. I did this since I thought password storing was the feature that contained the highest risk on our website. Furthermore, in my past projects, my team did not do a good job on this component and were hit by prank `SQL` Injections.[12] This catastrophic event occurred since my team did not value quality and therefore it was important for me to mention it.

[12] `SQL` Injection occurs when user input is not vetted for special `SQL` statements such as dropping all tables.

Conclusion

Throughout the term, I was able to learn about software engineering and deliver results to WEEF stakeholders. I learned how to use a repository to manage code with a team, significantly boosting productivity compared to my previous projects. I demonstrated the value of taking initiative by quickly eliminating massive dependencies such as a database migration. Furthermore, I took lead during retrospectives by creating a form to ensure everyone was reflecting on our work. I was able to solve real, complex problems for my stakeholder by breaking them down into smaller pieces. Previously, the WEEF website did not have search, which made basic tasks very painful. This was fixed using a scalable solution which adapted to dirty, real world data. Additionally, I implemented the key interaction that our stakeholder had with the website, which was editing funds. Lastly, throughout the project, I pushed my team to deliver quality. I wrote and created example code to handle errors and proactively searched for flaws in our system, such as in the way we stored passwords. Overall, the project was a success and I was able to learn valuable lessons that will scale into my future team-based endeavors.

References

Waterloo Engineering Endowment Foundation. About weef. URL http://www.weef.uwaterloo.ca/.