Debugging and Implementing new Features in a Production Codebase

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Andre Knocklein

Spring, 2022

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Rosanne Vrugtman, Department of Computer Science

Debugging and Implementing new Features in a Production Codebase

CS4991 Capstone Report, 2022
Andre Knocklein
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ak2qt@virginia.edu

Abstract:
A software company based in Northern Virginia had an expansive codebase in continuous development with the desire to fix customer-reported bugs and implement customer-requested features. The variety of tasks I was assigned to in a variety of languages using a variety of tools allowed for the opportunity to use a general debugging methodology. This involved the use of tools I had not used before like a Linux machine to host the local build and Google's site debugging tools. Using these tools allowed for an effective way to understand a codebase far too large to comprehend by traditional reading of the code. The methodology I used to understand the codebase allowed for the implementation of a dozen bug fixes and a few new features that are currently part of the product. The codebase still has bugs, and there will be a continuous maintenance of the code, as is the nature of production code.

1. Introduction

Production code is a monstrously large project even for those that have worked on it for a long time, so for someone with no experience with it, it can be challenging to understand. When I started my internship at SitScape, this became my problem. I saw hundreds of files of code with thousands of lines each while my largest coding project at that point was under a thousand lines long. I didn't know how I would manage to even understand it let alone work to improve it, but with the correct tools, the correct mentors, and enough time, I was able to do some very good work in that codebase.

2. Background

The problem was not just the size of the codebase, but also the way in which it was written. It was almost entirely composed of JavaScript and PHP which I had no experience with. Given that the codebase I needed to understand was in an unfamiliar language made its daunting size even more challenging to conquer. Another compounding factor was that there was almost no documentation for the code and certainly no centralized documentation. The only documentation that was available was conditional on whether or not the developer that coded that section decided to leave some comments on what they did. For an experienced programmer, this would not have presented much of a problem, but with little experience, I expected to struggle.

3. Literature Review

McCauley[1] (2008) is relevant to this paper as it discusses different ways that debugging occurs and also different ways of learning debugging. My experience at the internship was my debugging learning experience which can be contrasted with those in McCauley.

Similarly, Vessey[2] (1985) explores how both experts and novices go about debugging. This will provide a tool to compare my own

experience with so that I can place myself on a scale for before and after the internship.

4. Process Design

During my time at SitScape, I broadly worked on two different types of tasks. One was to fix bugs that have been reported; the other was to implement new features that will be useful to the customers.

4.1 Debugging

The size of the SitScape platform means that there are many opportunities for bugs to creep in. There is a process for finding these bugs and then fixing them which I went through often.

4.1.1 Problem Definition

Whenever we discovered a bug, we reported it to upper management who then created a ticket for it. This ticket was then assigned to me. Most of the time, the ticket itself did not give me a complete understanding of what the intended behavior should be, so I met with upper management to discuss exactly what was wrong and how it should work from a business perspective.

4.1.2 Cause Identification

Once I knew what the problem was, my first task was to identify where the problem originated. This is quite complex when the codebase is hundreds and hundreds of files with thousands of lines of code each which all work together to create the product. Sometimes, a problem seemed to have an easy-to-identify location in the code that caused the problem, but in actuality, the problem was caused by a line of code in a completely different file.

A big help in this was the use of a virtual machine I used in order to host the software locally. This allowed me full access to most of the files of the software through Google's debugging screen which allowed me to find the files containing the functions that get called when the problem is caused. Usually this just provided a starting point for my search for the cause, and I needed to keep searching by making use of breakpoints and inspecting values of variables. If a value was not what it should be, I used that as a road map to the location of the cause where I started debugging.

4.1.3 Debugging

Once I had found the cause, I needed to implement the fix. Usually, in the process of discovering where the problem was, some of the debugging and brainstorming of possible solutions was already done. The specific implementation of the changes I made varied widely among all of the tickets assigned to me, but the most common way in which I solved the problem was educated guessing with trial and error. Since I was new to the programming languages that were used, I used the rest of the codebase as a guide to come up with my solution. Usually, it would not work the first time around, and I had to go through many iterations. In this process, breakpoints became very useful as the nature of the codebase did not easily allow for print statements to readily show what was happening. Breakpoints were my favored alternative as the codebase was too complex to create test cases. Once I had implemented the change, I tried to recreate the bug, and I was always sure that any other functionality that might be affected also still worked.

4.1.4 Result Verification

When I was happy with my implementation, I usually showed upper management again to verify that they agreed that my implementation was satisfactory. This step did not always happen. For small bugs or if I was confident enough with my fix, I just pushed my git branch so as to not waste the time of my bosses.

4.1.5 Quality Assurance Process

Once my branch was pushed, it went into the quality assurance (QA) process. Here, a member of the QA team scheduled a

meeting with me to ensure that my solution worked as intended and more importantly that it did not break anything else. Because of this step, I felt it was okay to skip the Result Verification step.

When everything was cleared, the QA team member merged my branch into the development, and I could see my solution in the actual production code.

4.2 New Features

The process for implementing new features was often similar to that of debugging. The last two steps (Result Verification and Quality Assurance Process) are identical.

4.2.1 Requirements Elicitation

When I was tasked with implementing a new feature, the most important part was getting a complete understanding of the intended characteristics of that feature. This involved a meeting with my bosses in order to gain a list of requirements that the feature must have. This is a common step in any software development, and it was often helped by the fact that the codebase already had many features, many of them similar to what I had to implement, a fact that also became useful in the implementation. These meetings were usually lengthy as it was important to understand the requirements from the off so that there were fewer iterations once the actual implementation started. After the meeting was over, I had the responsibility of executing what my boss had tasked me to do in accordance with the requirements.

4.2.2 Implementation

With the requirements set out, my first task was to identify where to implement the feature. This was usually an easy task as there would be similar features already in place or my boss would tell me where they would like the implementation to be. After this, I would search the codebase for a function or piece of code that did something similar to what I wanted to implement so that I could adapt something instead of starting from scratch. This decision was driven by my limited knowledge of the programming languages I was being tasked to use, but it was actually very useful. It allowed me to both learn the programming languages but more importantly, it allowed me to understand the codebase better as my search for code to adapt would lead to a better understanding of the code I was adapting from.

As in the debugging process, the actual implementation was always varied, but the above process was consistent. When I implemented the feature, I made use of Google's debugger again and I used more breakpoints. The problem with my strategy of adapting other code was that there would be redundant or missing information. The majority of the time I spent implementing was to understand what information and which lines of code were truly needed. In some cases, I could adapt an entire function almost identically and just change one or two lines of code. Other times, the implementations were much more complicated, but the kit-bash approach I used was useful and allowed me to implement features quickly and robustly even with my lack of experience and knowledge.

After I had implemented the feature, the Result Verification and Quality Assurance steps were the same as outlined above.

5. Outcomes

Given the variety of fixes and features that I worked on, the outcomes of my work are similarly varied. My new features are used by both other employees at SitScape as well as the customers. The quality-of-life improvements I implemented through my bug fixes are especially useful to customers who need to learn how to use the SitScape platform. Some bugs used to

make some actions frustrating to work with, and after I finished my ticket, the customer did not need to wrestle with the software as much.

6. Conclusion

Working on these two types of tasks taught me a lot. Firstly, I learned a lot about JavaScript as that was the primary language I had to use to implement what I needed to. This will be very useful to me in the future as JavaScript is one of the most commonly used programming languages. I also learned some of the JQuery library for JavaScript which was also often used. Aside from JavaScript, I also now have a basic understanding of PHP. I did not use this much during my time at SitScape but the few functions I did write provided me with a basic understanding of how it works.

Maybe more valuable than the knowledge of the programming languages themselves is the learning how a production codebase works. This includes the tools used like Git and Virtual Machines, but it is more than that. Gaining experience with the way the system works in terms of how everything is put together is invaluable.

7. Future Work

The features I implemented and fixed are still active on the SitScape platform, and the platform is still being worked on. New bugs will emerge as the platform expands, and some features will not work well together which will cause problems. It is important to keep an eye on how all parts of the system interact so that problems can be identified. There will always be work to be done in improving the platform and fixing these problems.

References

1. Vessey, I. "Expertise in Debugging Computer Programs: Situation-Based versus Model-Based Problem Solving" (1985). ICIS 1985 Proceedings. 18. https://aisel.aisnet.org/icis1985/18
2. McCauley, R; Fitzgerald, S; Lewandowski, G; Murphy, L; Simon, B; Thomas, L; & Zander, C (2008) Debugging: a review of the literature from an educational perspective, Computer Science Education, 18:2, 67-92, DOI: 10.1080/08993400802114581