Maintenance vs Growth: Two Sides of the Same Coin

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

Eric 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

Brianna Morrison, Department of Computer Science

Maintenance vs Growth: Two Sides of the Same Coin

CS4991 Capstone Report, 2022
Eric Knocklein
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ek6ge@virginia.edu

ABSTRACT

A Virginia-based technology company with a long history sought to grow and modernize their service but was held down by legacy code and a lack of maintenance. Using concepts and best-practices gathered in my classes at UVA, I aided in both the growth and maintenance of these services by fixing bugs (modernizing the code) and implementing new features. I used concepts that are often seen as inconsequential to the development of programs: in-line documentation, descriptive variable names, modularization, etc. To achieve our goals for this task, the team and I had to communicate effectively both within and outside of the codebase. Though my project has only minimal outcomes to date, the outcomes are, nevertheless, crucial for the continued success of the product. Because both maintenance and growth are continual processes, future work includes adding structure to the codebase and increasing its modularity.

1. INTRODUCTION

What is more important: adding new features to a product or maintaining already-implemented features? This is a common question for many companies as they allocate their limited resources. Each has its own benefits, but maintenance is often neglected in favor of growth..

Growth is the flashier of the disciplines. It is usually much more satisfying for both developers and managers to create new features than to maintain old ones. New features draw in new customers with the promise of future prospects. On the other hand, maintenance is inherently a retrospective act. It serves to retain already-existing customers. Maintenance serves more than already-existing customers, though, because a poorly maintained software will not gain any new customers, either. Because of this, it is arguably more important than growth. However, the two disciplines cannot so easily be separated, for maintenance provides a foundation from which the program can grow. Growth, in turn, if done sustainably, can ease the burden of maintenance. Thus, growth is only viable in the long term if it is done sustainably and if the program is well maintained.

2. RELATED WORK

There is a plethora of research into software maintenance and debugging. Kaur and Singh (2015) name several key concepts in

reducing maintenance costs in software: re-documentation, elimination of cloned code, and, of course, the fixing of bugs, among others.

Bennett and Rajlich (2000) propose a model under which to view software development. In this model, my team was working on program evolution. According to the paper, this is the stage in which new features are added and in which the biggest structural changes occur, so it is where maintenance and growth coincide.

An early paper by Lientz, et al. (1978) emphasizes the importance of appropriate management of the maintenance phase of software development, citing it as the most crucial aspect of software maintenance and maintainability. My experience supports this, as it is difficult to write maintainable code or to maintain old code without some overarching vision of the structure and philosophy of that code.

3. PROCESS DESIGN

I was fortunate enough to experience both the maintenance and the growth of this platform. To understand the work that I completed, one must understand the systems under which I worked. During the discussion of these systems, which I will categorize into assignment systems and quality control systems, I will describe how they hindered or advanced my work. Working under such systems is crucial in a team in both the development of new features and the maintenance of existing features. While they may stifle some innovation or cause some friction in the development process, their benefits to organization are essential to the workings of the team.

3.1 Assignment Systems

Work to be done was assigned through tickets. Developers would receive a ticket with information about the bug they were supposed to remove or a feature they were supposed to implement.

Tickets varied in scope and difficulty with some simply requiring the changing of one line of code and others requiring the reworking of large portions of certain files. To reflect this, when a ticket was completed, the developer completed a "Level of Effort" field. This allowed management to gauge the workload of their developers and balance work among them.

To further aid in the balancing of the workload, tickets were assigned to projects, each of which had a few dedicated developers well versed in that area. This was not my experience, for I worked to fill in any gaps in the teams or on general tickets not well suited for a project assignment.

Interestingly, the assignment and submitting of tickets was done through two distinct web pages. While these offered almost the same functionality, they were presented quite differently and were optimized for certain functions. This decentralized form of communications was often cumbersome to use.

Other communication issues were also apparent in the assignment systems. While there were guidelines on ticket creation, they were rarely followed. The guidelines stated that a maintenance ticket should describe at minimum the element in which the bug resided, an image of the bug

or gap in action (or, preferably, a video), and a complete description of the current and desired behavior of the element in question. Instead, the tickets often only included one or two sentences that inevitably required lengthy meetings to clarify.

3.2 Quality Assurance Systems

Once the work to be done on a ticket is completed, it is sent to the Quality Assurance (QA) team. This team then verifies that the new behavior is correct and desirable. Because of the sub-par ticket writing, the desired behavior is not always clearly explained. As with the assignment of tickets, a meeting is usually scheduled to clarify the desired behavior, but this meeting is with the developer instead of the creator of the ticket.

It is required that every ticket pass through the QA team. This ensures the quality and consistency of the work done. While it does slow the process of development, it also prevents most faulty code from becoming embedded into the codebase. Such embedded code is much more difficult to revise than it is when it is first written.

However, there are some major limitations to the QA team, which does not assure the quality of the code (its readability or its style) but simply the outcomes of the code. This decreases the incentive for developers to write readable and maintainable code. No attention is paid to the specifics of a solution, either. Any solution that works is deemed correct. It is, of course, impractical to work through all the code that is written during the QA process, but simple steps can be taken to ensure the maintainability of the code.

3.3 Code Structure and Version Control

To understand the structure of the codebase, one must understand some details of the product being developed. It is a no-code website building tool whose interface is broken into a set of widgets. These widgets are linked to a node-based logical flow system, which handles the interactivity of the website.

The code is somewhat modular, with each widget or each node being defined in its own file. Within these files, though, the code ceases to be highly modular. Some functions are many hundreds of lines long. These functions composed files that were thousands or even tens of thousands of lines long.

As stated before, there was no real incentive for developers to write maintainable code, which could be part of the reason these files and functions were so cumbersome. Another consequence of the lack of coding standards was a lack of comments or in-line documentation. Legacy code, which only a few people at the company understood, dominated the codebase, leaving junior developers to spend hours attempting to locate whatever issue they were ticketed to address.

Some of these issues were alleviated by the version control and the relatively strict standards for interacting with it. The codebase was controlled through GitLabs, and each ticket was developed on its own branch. This gave each developer the freedom to experiment with solutions and implementations without fear of breaking

something else in production. Once a ticket was completed, the developer would write a merge request (MR) that detailed the changes made and the reason for those changes. In my experience, the writing of these MRs was much clearer than that of the tickets. This allowed developers to research the code they planned to work on before doing so, partly decreasing the issues with in line documentation.

3.4 Maintenance Work

The bulk of my work was on the maintenance of the codebase. For the most part, this was focused on finding and fixing bugs and inconsistency issues. To do this work, I had to gain an understanding of the codebase, which was sometimes challenging because of the details provided earlier in this report.

It is often more difficult to parse another developer's solution to a problem than it is to invent one. I found this to be true in my work, especially when there are very few comments and no external documentation.

3.5 Novel Work

While not as frequent as the maintenance work, my work in adding new features represents some of my biggest contributions to the company. Writing a solution from scratch is often simpler than revising a solution that is already implemented. As such, being able to add new features offered some freedom to write code the way I saw fit with the standards I deemed necessary.

4. ANTICIPATED OUTCOMES

Being just an intern at an established company meant that my work had a fairly small effect on the company as a whole. However, my work has highlighted flaws in the company systems and is still being reviewed and utilized by the company.

I anticipate that my code and the issues that I raised to management about the structure of the code will drastically improve the maintainability of the codebase, which will have spillover effects such as reducing the work needed to train new developers. This last point is especially important for growing companies such as this.

5. CONCLUSION

Part of my goal with my maintenance work was to improve the readability of the already-existing code. However, within a complex function or program, it is difficult to be confident in one's understanding of code written by someone else. As such, I was rarely confident in my ability to document code that was not my own. This severely dampened the impacts of my work. Even so, my code added some much-needed modularity and stability to certain sections of the code. The same is true for the few features that I implemented. Not only are these features necessary, but they are also written in a readable and maintainable manner.

6. FUTURE WORK

Since growth and maintenance are both continuous processes, there is still much work to be done. The most crucial shortcomings of the system are outlined in this report, and these are the areas that should be addressed first. However, most of

these changes must be implemented broadly, as they are managerial in nature.

The code itself is quite mature, but much of it is burdened by legacy code. A crucial next step is to modernize the codebase, and this is what I will be working on in the coming months. The team I have been placed on is working to integrate React into the codebase to make the company's webpages much more reactive and user-friendly. This comes along with changes in the User Interface (UI) styling. Changing this core system of the website will likely expose other shortcomings that can be fixed in this new reactive and modern manner.

The structure of React also lends itself well to fixing the shortcomings listed in this report. It is built to be modular and encourages modular design. Without functions that are thousands of lines long, the code will be made much simpler and more readable. In turn, inserting comments will be made less cumbersome because each comment has to describe fewer complex behaviors. Hopefully, this will lead to individual developers being encouraged to write their own readable and well-documented code.

7. UVA EVALUATION

I could not have completed the work as efficiently as I did without the guidance and experience provided to me by my UVA coursework. Though it did not provide me with experience in the specific tools I needed for the job (JavaScript and jQuery), it gave me the tools necessary to learn what I needed on my own. There is a large emphasis placed on learning my doing in the UVA CS department, which fits well with my work experience. There are no lectures to watch or tutorials to follow (even though the documentation should have, perhaps, provided such resources); I learned by doing.

The most analogous class to this work environment was CS 3240 Advanced Software Development. While this class should have provided a wealth of pertinent experience and applicable knowledge, this was not the case. The only true applicable skill learning in the course that was used in the internship was how to use git effectively. Even that, though, I learned more effectively and at a greater level than during my coursework.

What hindered my learning (and what likely hinders the learning of many other students in the course) is the group-work oriented structure of the course. There is a fundamental disconnect between how group work functions in academics compared to the way teamwork works in the workplace. This disconnect stems from the differing motives of the individuals participating in each system. Workers who do not work do not get paid, while students who do not work are carried by their teammates. This incentivizes students to offload their weight and responsibility onto the group-member that is most diligent, severely hindering their own learning as well as that of their teammates.

It would provide a more analogous work environment if students were able to choose their teammates. This would provide teams with similar levels of commitment and an enhanced ability for accountability

through social pressures (as would exist in the workplace).

REFERENCES

[1] Kaur, U., and Gagandeep S. “A Review on Software Maintenance Issues and How to Reduce Maintenance Efforts.” International Journal of Computer Applications, vol. 118, no. 1, 2015, pp. 6–11., https://doi.org/10.5120/20707-3021.

[2] Bennet, K. H. and Rajlich, V. T. 2000. “Software Maintenance and Evolution: a Roadmap” Research Institute for Software Evolution Department of Computer Science University of Durham Wayne State University UK Detroit, MI 48202 DH1 3LE USA

[3] Lientz B. P., Swanson E. B., and Tompkins G. E. 1978. Characteristics of application software maintenance. Commun. ACM 21, 6 (June 1978), 466–471. https://doi.org/10.1145/359511.359522