Addressing Challenges in Software Engineering

(Technical Paper)

Code Maintenance and Deprecation: Why is Code Deprecation Necessary?

(STS Paper)

A Thesis Prospectus Submitted 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

Morgan Kinne

October 27, 2023

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

Signature _____ Date_____

Morgan Kinne

Approved _____ Date_____

Brianna Morrison, Department of Computer Science

Approved _____ Date_____

Richard D. Jacques, Department of Engineering and Society

Introduction:

Between June and August of 2023, I worked as a site engineering intern at the Washington Post, a newspaper company with an online platform. Because of the company's massive online platform, the overall engineering department was split up into sections and teams. I worked on the Site Core Engineering team, a division of the Site Engineering team whose main purpose is to support the website. The Core team's responsibilities included backend software engineering, which involves working on server-side software, or anything that can't be seen on the website. The team's daily work involves maintaining and supporting the website from behind the scenes, and working with different teams to develop and update the code that helps the website run efficiently and effectively. My technical report will focus on the purpose of my work as an engineering intern and the challenges I faced in that position. I will dissect the experience I had of developing an Application Programming Interface or an API. An API is often described as a web service, which is software that sends data and provides a service to the website. The purpose of redeveloping this API was to replace an older API that had become outdated, addressing issues with software maintenance, especially on older products. Software maintenance refers to upkeep to make sure that the software is updated and runs as expected. My STS research will focus on issues of code deprecation and maintenance and the impacts of aging software on engineers and their teams. Code deprecation is defined as the process of marking a software system or service as deprecated so that it is not used to support other software and is essentially phased out. Code maintenance is a vital part of software engineering, and when done properly, it can eliminate excessive time-consumption on engineering teams and can improve the dependability and quality of the code developed by those engineers. In this paper, I will discuss the technical challenges I faced at my internship, the sociotechnical implications of those

challenges and the research done on them, as well as possible solutions which explore the issue behind failed code maintenance.

Technical Discussion:

In today's world, software development is expected to run smoothly and efficiently. This includes the expectation that onboarding new developers onto a team will not take up too much time. The specific project that I worked on this summer was an API that had been marked as deprecated because of several problems. Alongside issues such as a lack of documentation and convoluted error handling came the time-consuming task of onboarding new developers onto the outdated platform, which slowed work on other projects. Thus, the decision was made to develop a new API in Python. Beyond a simple rewrite, this project would include extensive testing and documentation to prevent the new API from becoming obsolete. Collaboration on this project included another intern and me, as well as a senior developer who oversaw our work. I reached out to developers who had initially worked on or with the older JavaScript API, to affirm that our new API would be compatible with their previous work. The scope of this effort included not only addressing issues related to maintenance but also optimizing the design for long-term use. Noteworthy components of the project included working in AWS to store data important to the API, utilizing the Flask web framework, and working in Docker to deploy and test the application. Alongside this work came strict code style adherence, which was accomplished using the Git hooks and pre-commit libraries. During my internship, the primary goal was an efficient and easy to understand Flask API that would replace the old API. The project aimed to create a seamless integration between the API and various external services, incorporating AWS SSM, client APIs, JSON data parsing, image processing, and storage using AWS S3. The initial phase involved setting up a GitHub repository based on a Flask API template. A crucial component of this setup

was the implementation of the config.py file. Using AWS SSM, I designed a configuration system that securely retrieved credentials for accessing the client interface. These credentials were then made available to the software, ensuring secure communication with the client APIs. Building upon the configuration setup, I developed a Python software component that interacted directly with the client APIs. These APIs provided JSON data, which I parsed and processed for storage within the Flask API. The process involved understanding the JSON structure and designing efficient algorithms for data parsing. A key aspect of the project was ensuring code reliability and maintainability. To achieve this, comprehensive unit tests were created for the JSON parsing logic. Simultaneously, I documented the codebase, providing clear explanations of functions, methods, and modules. This documentation facilitated easy understanding for future developers and team members. The Flask API endpoints were carefully designed and implemented to handle incoming requests efficiently. Each endpoint was tailored to specific functionalities, ensuring modularity and scalability within the API architecture. This step involved a deep understanding of the API's core requirements and client expectations. Maintaining consistent code style was crucial for collaborative development. I integrated the git pre-commit hook library, ensuring that the code adhered to predefined style guidelines before each commit. This automated process streamlined code reviews and enhanced code quality. One of the complex tasks involved parsing images from the JSON data, converting them into JPEG format, and storing them securely in AWS S3. Additionally, I implemented a Content Delivery Network (CDN) to optimize image delivery, enhancing the overall user experience. Each image was versioned, allowing regeneration and updates as needed. The final phase involved rigorous testing of all integrated components, ensuring seamless functionality and error handling. I created extensive test cases for various scenarios, validating the API's performance under different conditions. Concurrently, I added detailed

documentation for the entire development process, including setup instructions, API endpoints, and usage guidelines. In summary, the project design encompassed a systematic approach to API development, focusing on security, efficiency, and scalability. The integration of external services, thorough testing, and comprehensive documentation were pivotal in ensuring the successful completion of the internship project. At the end of the internship, the API was fully developed with completed unit testing and documentation. I deployed the API before leaving the internship. With the finished product, there are several implications to consider. Proper maintenance of the API will help reduce the chance of the API becoming outdated again, and with the documentation and testing included, maintenance should be a simple task for the team to take on. The API was also built to consider any dependencies. For example, the front-end side of the website relies heavily on the structure of the information that is returned by the back-end API, so we needed to ensure that the delivery system would be the same. Ultimately, the completion of this API not only resolves the issue of onboarding future developers, but it also serves to remind of the value of planning and revitalizing aging software.

STS Discussion:

My STS research will focus on code maintenance and its impact on software engineering teams, specifically relating to how there are many dependencies between APIs and the web services they support. J. Yasmin et al. (2020) explore the direct impact on dependent applications when APIs are not properly maintained. This paper delves into the proliferation of web APIs and how the dynamic nature of APIs poses a fascinating sociotechnical challenge. APIs usually serve as something that can show how software development is conducted, governed, and experienced. This paper talks about how the expansion of web APIs, as rapid as it was, demonstrates the intertwined relationship between technological advancements and sociotechnical dynamics. Web

APIs, designed to facilitate seamless connectivity, create intricate networks of dependencies between software systems, data, and algorithms. These dependencies can also introduce vulnerabilities and uncertainties, which can impact both API developers and users. The lack of standardized protocols and the inherent change-proneness of web APIs challenge the stability and predictability that developers rely on. Moreover, the deprecation of API elements illuminates the negotiation of technical change within sociotechnical systems. The process of deprecation involves not only altering code but also renegotiating social contracts between developers, API providers, and consumers. In the absence of clear standards for deprecation practices, developers end up navigating a landscape where technical decisions intersect with social practices, ethics, and communication protocols. J. Yasmin et al.'s study highlights the discrepancies where a significant number of APIs introduce breaking changes without proper deprecation warnings. This generates challenges and delays for engineers tasked with maintaining applications built upon these APIs. By delving into the sociotechnical dimensions of web API deprecation, this study contributes to the technical understanding of software maintenance and offers insights into the connection between technology and society. It sheds light on the sociotechnical challenges faced by developers and API stakeholders, emphasizing the need for standardized practices and enhanced communication channels. D. Ko et al. (2014) analyzed how deprecated APIs, if not replaced promptly, can result in security vulnerabilities and performance issues, which emphasizes the importance of timely maintenance in APIs to avoid negative impacts to other applications. Considering this study and drawing parallels with my own experiences from this summer, it becomes evident that the quality of API documentation plays a pivotal role in the maintenance and resolution of deprecated APIs. The rapid evolution of software libraries emphasizes the need for developers to stay updated. However, the process of updating

applications to accommodate these changes becomes intricate when API elements are deprecated, especially if the associated documentation lacks clarify and completeness. Insufficient API documentation, a problem I and my team experiences firsthand, hampers the ability of developers to properly maintain code and identify suitable actions. This lack of guidance not only results in unreliable applications but also poses security risks and performance issues, as highlighted in this text. The study's findings reveal a stark difference in the resolution of deprecated APIs based on the presence of documentation. In my own experience, encountering inadequately documented deprecated APIs led to extensive delays in my development process, emphasizing the real-world consequences of incomplete documentation. Hence, this study reinforces the critical need for API providers to prioritize not only deprecation warnings but also clear, informative, and easily understandable documentation to facilitate seamless transitions for developers, ensuring the robustness and security of software applications in the face of evolving APIs. A. A. Sawant et al. (2018) seeks to understand the reasons behind API deprecation and how crucial it can be to the development lifecycle. They emphasize how the lack of clarity on deprecation resources can lead to confusion and delays as developers may not know how to effectively react without further information, which is something I will analyze, looking at the reasons the old version of the API I developed this summer was deprecated. To expand on this, my research will look at the findings from this study to understand the reasons that API producers choose to deprecate features. This study proves how vital is it for API consumers to make informed decisions on how to react to an API that is no longer fully functional. However, it also highlights how while deprecation, though a common practice, is far from straightforward. It involves a nuanced decision-making process by API producers, often influenced by various factors such as evolving user needs, technological advancements, and

changing software landscapes. My research endeavors to explore the implications on software engineering teams, particularly focusing on the developers tasked with maintaining applications reliant on these APIs.

In summary, my research will bridge the gap between API producers and consumers, shedding light on the intricate dynamics of API deprecation. By understanding the motivations behind deprecation and the challenges faced by software engineering teams, we can pave the way for more effective strategies in code maintenance, ensuring smoother transitions for developers and, ultimately, enhancing the overall stability and efficiency of software engineering processes.

Research Question and Methods:

This research is driven by a fundamental inquiry: how do software engineering teams navigate the intricate challenges posed by code maintenance and deprecation? Specifically, this study seeks to unravel the complexities inherent in managing legacy code and devising effective strategies for updating deprecated code or APIs. To delve into this field of multifaceted challenges, I will analyze existing literature and empirical studies, and I will aim to distill valuable insights that can inform innovative solutions within the field software engineering. While exploring this literature, I will strive to find effective strategies that teams can use as well as the core problem behind failed code maintenance.

A substantial portion of this research endeavors to dissect the existing body of literature and empirical studies dedicated to code maintenance and deprecation. This extensive literature analysis serves as the cornerstone of my inquiry, allowing me to navigate through the wealth of knowledge amassed by software engineers and researchers. I aim to extract invaluable insights,

distilling them into actionable strategies that engineering teams can employ. Furthermore, this deep dive into literature will unravel the core problems that often lead to the failure of code maintenance efforts. By identifying these underlying issues, I hope to contribute nuanced perspectives to the ongoing discourse within the software engineering community.

Beyond merely summarizing existing research, my aim is to identify patterns and discern best practices that have proven effective in real-world scenarios. This involves a meticulous analysis of the literature, examining the strategies employed by teams that have adeptly managed legacy code and deprecated APIs. By understanding these success stories, I intend to unearth the underlying principles that can serve as guiding beacons for other engineering teams facing similar challenges. Additionally, the identification of recurrent pitfalls and challenges will shed light on the hurdles that demand innovative solutions.

A critical facet of this research involves dissecting failed code maintenance endeavors. By scrutinizing cases where software engineering teams encountered insurmountable challenges, I aim to uncover the fundamental issues that impede successful maintenance efforts. This analysis of failure is integral to the learning process, providing invaluable lessons that can inform future strategies. By unraveling the root causes of failure, this research seeks not only to highlight challenges but also to pave the way for proactive solutions, ensuring that engineering teams are equipped to navigate the complexities of code maintenance and deprecation successfully.

In this study, I will synthesize these insights, weaving together a comprehensive narrative that explores the multifaceted landscape of software engineering challenges. Through this rigorous inquiry, I aspire to contribute substantively to the ongoing discourse, fostering an

environment where innovative solutions and informed strategies thrive, ultimately advancing the field of software engineering.

Conclusion:

In conclusion, the successful resolution of code deprecation challenges holds the key to fostering innovation and sustainability in software engineering. By addressing the technical intricacies and sociotechnical dimensions of code maintenance, developers can pave the way for seamless transitions from outdated systems to modern, efficient solutions. The envisioned technical deliverable, a meticulously developed API, coupled with the sociotechnical insights gained from this research, has the potential to significantly contribute to the amelioration of problems associated with aging software. Through this comprehensive approach, the software engineering community can embrace the future with confidence, ensuring the continued evolution of digital technologies.

References:

A. A. Sawant, G. Huang, G. Vilen, S. Stojkovski and A. Bacchelli, "Why are Features Deprecated? An Investigation Into the Motivation Behind Deprecation," 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, 2018, pp. 13-24, doi: 10.1109/ICSME.2018.00011.

D. Ko, K. Ma, S. Park, S. Kim, D. Kim and Y. L. Traon, "API Document Quality for Resolving Deprecated APIs," 2014 21st Asia-Pacific Software Engineering Conference, Jeju, Korea (South), 2014, pp. 27-30, doi: 10.1109/APSEC.2014.87.

J. Yasmin, Y. Tian and J. Yang, "A First Look at the Deprecation of RESTful APIs: An Empirical Study," 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, SA, Australia, 2020, pp. 151-161, doi: 10.1109/ICSME46990.2020.00024.