Self Service Tools and its Impact on Business Efficiency

(Technical Paper)

Emphasis on Software Maintenance in High Priority Platforms

(STS Paper)

Thesis Prospectus

Fardeen Khan

School of Engineering and Applied Science, University of Virginia

STS 4500: STS and Engineering Practice

Dr. Richard Jacques

October 27, 2023

Introduction

I worked as a Software Engineering Intern at Capital One from June 2023 to August 2023. Capital One is a consumer credit card and bank holding company and many of its functions are driven by big data and cloud-based solutions. The technical topic of this report will provide an overview of the skills I learned pertaining to software engineering within a team in a larger organization and the adjustments needed after having experience with primarily smaller and more centralized teams. It will also reiterate how they were used in the making of a self-service tool to streamline business operations by automating the process of tracking enterprise messages sent to customers. The STS Research portion of this report will present over how software is maintained in these larger codebases and the underlying impact the software development lifecycle has on developments of future systems built on top of it; frameworks and libraries used in large-scale production applications will likely have incremental upgrades, and those applications will need to be migrated to newer versions to ensure security, performance, and persistence while also maintaining a functional state for the application.

Technical Discussion

As a computer science student since high school, I have been exposed to many different group projects that include building a system from the ground up. However, these groups have been made up of a few other students working together and have similar technical capabilities. However, I learned that, throughout my time as a software engineering intern, there is a much

larger group of people with whom I must be able to interact with. The specific product I was working on was part of a larger suite of tools the company uses to manage and analyze all its messaging capabilities. Essentially, there is a database containing metadata about each message such as its delivery status, unique id, language, message type code, etc. As a result, being a newcomer and having a relatively short onboarding time with the platform and its related components, I had to learn much of it on the job and with the help of the other engineers, as the project entailed the use of Vue.js front-end framework, Java Spring API framework, cloud pipeline tools such as AWS, Jenkins, Docker along with Capital One's internal managed pipeline for deployment. Additionally, amidst my work on my team's assigned project, the platform was also set to undergo a migration from Node version 14 to Node version 18, contributing to my experience in keeping a software application maintained and up to date over the course of this 10-week internship. This report will be written following guidance from the CS 4991 teaching department, specifically Brianna Morrison.

A key component of my project deliverable is its user interface, as it is the main form of interaction a user will have with the service. As a result, it was imperative the design of the interface complied with the "User Interface Guidelines" set for all the company's software to maintain a streamlined experience for all users. My development team was assigned a design team to collaborate with on the user interface and ensure these guidelines were followed with the app. This meant certain icons, error messages, fonts, symbols, etc. were to be used when designing it.

The design team also specified the necessary information to be displayed on the page when returning a result: the send timestamp, message language, message id, delivery status, delivery method (mail, push notification, call, etc.), message type. These specific data fields were chosen because they were deemed the minimal necessary fields but would still provide ample information to debug a messaging failure scenario while keeping the visual state clean.

Implementing a tool to track message statuses requires a three-tiered approach like most apps: a front-end page developed using Vue.js framework, a backend logic controller configured using Java Spring Boot, and the data layer, which is hosted on Amazon's AWS DynamoDB service for this product.

Due to the structure of our summer internship, the front-end component was the first step towards creating the app. The goal for the user interface was to have a search bar upon the page loading for the first time. The search bar only accepts a valid message status ID, which will then be used to look up a matching entry in the database and return its associated data fields. The fields are then displayed in a 3x2 grid. If the input into the search bar is not a valid ID, there will be an error message shown to the user saying there must be an 18-digit message ID entered. There will also be errors communicating application issues, such as HTTP 3xx, 4xx, and 5xx statuses. The exact message was drafted with the design team to adhere to the company's UX guidelines.

The majority of this app's functionality was spent on the back-end services, using Java Spring Boot. During the process of sending a message to the users, the message is sent to a message vendor, such as Apple or Google for push notifications or SparkPost for other forms of

communication. These vendors return status codes showing the status of the message. Additionally, as the message is sent, the messaging system also stores data of its own, such as timestamps, message types, message size, transfer rate, language codes, etc. The API would be used to access this data from the database upon request from the front end.

Specifically, the main function to be implemented is the one retrieving the necessary data fields. The format of the returned data is required to be a JSON format map, where the fields, send timestamp, message language, message id, delivery status, delivery method (mail, push notification, call, etc.), message type, would map to their values. This JSON payload would then be sent to the Vue app to be parsed into the HTML page.

The reasoning for this design is mainly for the purpose of resiliency and speed. Using Java allows for a broad range of contributors and Spring Boot is available as a very scalable framework, which is necessary for a platform supporting over 66 million daily messages to be sent while avoiding overloading the infrastructure. Vue.js is a modern front-end framework that can combine the advantages of both Angular.js and React.js allowing for simplicity while developing an app.

Throughout the development pipeline, there are three stages used to deploy versions of the application to. Deployments must move from development environments to QA environments to the production environment, allowing each to test the state of the app as new features are added. Jenkins CI is used to automate this process, along with the company's internal managed pipeline, which reads a YAML file containing configurations for the infrastructure using AWS and

Docker. Specifically, the YAML file contains the necessary IAM roles for the app's AWS services to access the data. It also contains information for starting up the ECS Fargate service and initializing a Docker container with it using the provided Dockerfile. Once the ECS stage is complete, the app is deployed into multiple regions for resiliency, along with traffic routing for load balancing. As a result, a sudden increase in the traffic would not immediately exhaust one region, but would route traffic to another to ease the load on it. Also, failover capabilities are implemented to allow for the resilience, so the app doesn't lose its functionality in case there is a failure on one of the regions.

The stated purpose of this tool is to alleviate stress off engineers while still allowing clients to easily view and debug their problems related to sent messages. Upon deployment of this application into production environment, it is available to all internal employees with a valid message id, allowing them to quickly see the status of their desired messages.

Prior to this app, engineers would spend on average one hour a day searching through a database to present the clients with their issues and potential solutions. Because the app is a self-service tool, it can operate autonomously, relieving engineers of this duty while also presenting the clients with immediate results, saving time for both parties.

STS Discussion

A crucial aspect in the lifetime of most large scale applications is the period after development and deployment of a Release Candidate version into the production environment – maintenance until its decommission. From an initial business standpoint, it may seem more

beneficial to direct support towards the development of a new platform. However, considering that most of the code and software often serve multiple other platforms, keeping the underlying foundational pieces of software is imperative to system reliability and resilience.

According to Lientz at al (1981), maintenance does demand copious portions of a developer's time, as roughly half an application's time is spent on its maintenance and conversely a developer can be expected to contribute half their time to software maintenance, and Bennett et al (2000) provide insights into where efforts should be focused during a software application's lifetime. They mention that there is thought given to potential design changes and feature requests by the users after the product has been delivered as incremental upgrades, however, most of the changes they get are ones that are not thought of during design time. These changes can be in the form of necessary changes to the system environment, user requirements, error handling, or preventing anticipated problems. Largely, a changing system environment is inevitable, as keeping software updated is not only crucial for long-term support of an application, but also for modernizing performance and sustaining security levels as newer infrastructure is rolled out.

Another important aspect of collaborative software engineering is managing turnover of developers on a team. As such, experienced developers who have in depth knowledge of an application take with them their knowledge when they leave, and the new developers are onboarded by the remaining team. Etamadi et al (2021) explore the best possible methods teams may use when onboarding new contributors.

Possible mitigations to the knowledge loss in an era where employee turnover is common are knowledge transfer documents and documentation about a specific part of the team's working suite, commenting code, or modelling and diagramming the underlying structure of the system. However, it is possible that newer, especially junior, developers can overlook these documents.

Thus, potential solution to the problem caused by one or more relatively inexperienced member of a development teams is to assign them tasks that are specifically not in their knowledge zone, as this will enforce the developers to consult other, more experienced, members of a team to assist. While possibly leading to longer times to complete a task, Etamadi claims the term "knowledge diffusion" to the result of this scenario, as the more experienced developers will leave the inexperienced developer with more information than they previously had. On the other hand, assigning tasks to developers who are already proficiently knowledgeable about the issue assigned will result in shorter times to complete a task and more efficiency and in turn less costs. However, doing so will also lead to a concentration of knowledge in one individual, which is detrimental to the overall team if the developer is no longer a part of the team. Upon a newer developer joining the team, it may be beneficial to diversify their range of tasks to give them a broader view of the platform they are working on, and once a proficient understanding of the overall codebase is achieved, they may start being assigned more specialized roles.

Conclusion

Self-service tools such as this are believed to provide value to a business by automating the process of handling and accessing information, making it less dependent on other employees. Development of this, like any other application requires consideration of the architecture of how

the front-end, back-end, and data layer interact with each other in the company's existing tools. Along with this, attention must also be directed to the support of the product down the line in the form of performance updates, feature additions, and security patches.

Bennett, K., & Rajlich, V. (2000). (rep.). *Software Maintenance and Evolution: A Roadmap*. New York, NY: Association for Computing Machinery.

Etemadi, V., Bushehrian, O., & Robles, G. (2021). (tech.). Task assignment to counter the effect of developer turnover in software maintenance: A knowledge diffusion model.

Lientz, B. P., & Swanson, B. (1981). (rep.). *Problems in Application Software Maintenance*. New York, NY: Association for Computing Machinery.