Legacy System Modernization at CarMax

(Technical Paper)

Perspectives, Challenges, and Techniques Around Legacy System Modernization

(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 of the Degree
Bachelor of Science, School of Engineering

Matthew Y. Samuel

Fall 2023

Signature ____ Date __________

Matthew Y. Samuel

Approved ____ Date __________

Briana Morrison, Department of Computer Science

Approved ____ Date __________

STS Advisor: Richard D. Jacques, Ph.D., Department of Engineering & Society

Introduction

Organizations and businesses are today using many legacy system platforms nationwide. These platforms are systems that contain outdated software and/or hardware, and they are still being used since they fulfill their intended purpose. However, with how fast business needs, consumer needs, and software are evolving now, maintaining legacy system platforms is not viable for multiple reasons. First, these systems are expensive to maintain due to technical debt increasing the maintenance costs for these platforms. Also, these systems are inefficient, which impacts employee productivity and create a negative experience for the customer in some way. Finally, these legacy system platforms are not compatible with newer systems.

Between June 2023 and August 2023, I worked as a Software Engineering Intern at CarMax, the nation’s leading used car retailer. My technical report will examine the work that I did and the challenges that I faced during my time at CarMax. More specifically, I will focus my technical report on my experience developing a microsite and microservice that will allow CarMax to modernize an existing in-store legacy system platform that enables sales managers to perform the final approval of customer-submitted stipulation documents. My STS research will focus on the various techniques that organizations can utilize to modernize their legacy system platforms. Many organizations still have legacy code in their software, which has many downsides, including, but not limited to, weak security, incompatibility with other software, inefficiency, and high maintenance costs.

Technical Discussion

During my internship at CarMax, I learned a lot about software engineering and working in a cross-functional team. The web application and backend microservice that I worked on developing alongside four other software engineers and one other intern is developed using the

ReactJS framework that uses the JavaScript programming language for the frontend, the ASP.NET Core framework that uses the C# programming language for the backend, and the site is hosted in the cloud using Microsoft's Azure App Services. I gained experience in web development, deploying web applications, and continuous integration. Additionally, I was able to work in a cross-functional team of software engineers, designers, product managers, quality engineers, and delivery managers. This taught me how to effectively communicate with others who do not have the same technical background as I do, a skill that will be crucial for the rest of my career. Furthermore, I learned about legacy systems, including the dangers of maintaining these systems and the importance of modernizing these systems. The internship lasted 10 weeks, and I modernized the legacy system platform for the entirety of the internship.

To modernize this legacy system, I utilized various web frameworks, programming languages, and tools to develop a minimum viable product (MVP) that enables sales managers to approve stipulation documents from within an existing site that allows associates to do the initial approval of these documents. This new sales manager experience consists of multiple components: the stipulation card stepper page, the document viewer, and the final submission confirmation modal/popup and submission confirmation page. The MVP consists of a microsite front-end developed using the ReactJS web framework and a microservice on the back-end developed using the ASP.NET Core framework and the C# programming language. The web application is hosted using Microsoft Azure App Services.

The stipulation card stepper page consists of multiple sub-components. First, it contains order details, including the vehicle stock number, the primary buyer's name, the co-buyer's name (if applicable), the store number, and vehicle information, including year, make, and model. Additionally, this page includes one card for each stipulation type (e.g., Proof of Identity, Proof of Income, Proof of Insurance) that can be expanded to show the specific documents that are required for that stipulation type (e.g., Pay Stub for Proof of Income).

From here, the sales manager can click on a specific document within the stipulation category, which will take them into the document viewer, where they can navigate through the files within that document and verify additional information about the buyer, including, but not limited to, income, contact information, and address information. When the sales manager returns to the stepper page from the document viewer, they can click the "Verify Stipulation" button to confirm that these documents will satisfy the requirements for the particular stipulation type.

Once the sales manager has verified documents for all stipulation types required for a particular vehicle order, they can click the "Submit to Business Office" button at the bottom of the stepper page, which will prompt them with a popup asking if they are sure they want to submit the order, because after submission they will no longer be able to edit it. If the sales manager clicks "Submit" on this modal, they are routed to a confirmation page, and all the input boxes on the stepper page and document viewer become read-only. There is also a similar process in which the sales manager can click the "Decline" button to decline the order and blacklist the customer.

The front-end of this microsite was developed with the ReactJS framework, which utilizes an extension of the JavaScript programming language called JSX that allows HTML to be embedded within a JavaScript file. Within the front-end of the site, there is a back-end that adheres to the Backend for Frontends (BFF) pattern, which is a variation of the API Gateway pattern. This pattern creates a separate API gateway for each type of client (e.g., web app, mobile app, etc.), which allows for separation of concerns and easier maintenance as the size of the application grows.

The back-end of this microsite, also known as the microservice, was developed with the ASP.NET Core framework, which utilizes the C# programming language. The back-end adheres to the Orchestrator Pattern, which involves the controller, orchestrator, service, and repository layers. When the user interacts with the UI of the site, input is first sent to the controller through the BFF layer. From there, it is passed onto the orchestrator layer through a Data Transfer Object (DTO) that is used to move data between the UI and the API. The orchestrator calls the service layer and passes it data from the DTO, and the service layer contains business logic such as input validation, grabbing model objects from the repository layer, and modifying these objects and sending them back to the repository layer to update records in the database, if necessary. Finally, the repository layer takes in data from the service layer and retrieves and/or updates data from the database.

This web application is hosted on the cloud using Microsoft Azure App Services, where the resources are automatically scaled up/down and the load balancer distributes network traffic among multiple servers, both done to ensure an extremely high rate of availability for the site.

Kazanavičius and Mažeika (2019) define a microservice architecture as "suites of independently deployable services organized around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data." Since the monolithic legacy system is broken down into smaller services that communicate with each other seamlessly, maintaining this web application that uses the microservices design architecture will be easier than maintaining the monolithic legacy system platform currently being used in stores, especially since various teams can develop and maintain these services independently. Another benefit of this microservices architecture is that these applications will have better scalability, especially if they are deployed on the cloud.

STS Discussion

There are differing perceptions of legacy system platforms and their modernization between people in academia and people in industry. This includes perception of the definition of a legacy system platform, benefits of legacy systems, and reasons to modernize legacy systems. For example, based on research conducted by Batlajery et al. (2014), most participants agreed that a legacy system is an old system, but some participants had different definitions, including “systems and technologies that do not belong to your strategic technology goals” (Batlajery et al., 2014). Additionally, more than half of participants believed that a programming language was not an indicator of whether a system was a legacy system, while the rest of the participants agreed that it was an indicator.

Furthermore, the modernization of legacy system platforms presents organizations with many challenges. Some of these challenges include complex system architecture, lack of knowledge, data migration issues, pushback from leadership within the organization, and time constraints (Batlajery et al., 2014). Although the modernization of legacy system platforms has different meanings to different people and comes with many challenges, it is still a necessary process for organizations to undergo so that they can meet the rapidly changing needs of their customers, their business, and the market.

Therefore, my STS research will aim to examine what a legacy system platform is, the importance of modernizing legacy systems, and the various techniques that organizations can use to modernize their legacy system platforms. Fanelli et al. (2013) define a legacy application system as a system made up of old software and/or hardware, and they go on to define a framework for modernizing these legacy systems that organizations can utilize. I would like to explore various methods for modernizing these platforms because “researchers estimate that 180-200 billion lines of legacy code are still in use today” (Fanelli et al., 2013), which means that maintaining and modernizing this software is crucial, especially as business needs, consumer needs, and software evolves rapidly. Chiang and Bayrak (2006) present a semi-automated method that businesses can use to convert legacy code to components in such a way that they can communicate effectively with each other. Furthermore, I would like to investigate the pros and cons associated with various modernization techniques. Comella-Dorda et al. (2000) examine a plethora of modernization techniques that fall under one of three categories: user interface modernization, data modernization, and functional (logic) modernization. Understanding not only how to modernize legacy system platforms, but also the pros and cons associated with each modernization technique, will enable organizations to succeed in their modernization efforts.

Conclusion

The microsite and microservice that I developed at CarMax this past summer is an example of the modernization of a widely used legacy system platform within an organization. The front-end of the microsite was developed with the ReactJS framework using the JavaScript programming language, and the back-end microservice was developed with the ASP.NET Core framework using the C# programming language. The perception of legacy system modernization is different between academia and industry, and the modernization process introduces many challenges. Therefore, my STS research aims to examine what a legacy system platform is, the importance of modernizing these systems, and the various techniques organizations can use to modernize their legacy systems, including the pros and cons associated with each technique. This research is important because many organizations struggle to modernize their legacy systems, but the longer they wait, the more issues it will create. Therefore, there must be more education on the topic and process of modernization to help these organizations in their modernization efforts, which will in turn allow their systems to be more secure, better integrated with other systems, and more easily maintainable, among other benefits.

References

Batlajery, B.V., Khadka, R., Saeidi, A., Jansen, S., & Hage, J. (2014). Industrial Perception of Legacy Software System and their Modernization.

C. -C. Chiang and C. Bayrak, "Legacy Software Modernization," *2006 IEEE International Conference on Systems, Man and Cybernetics*, Taipei, Taiwan, 2006, pp. 1304-1309, doi: 10.1109/ICSMC.2006.384895.

Comella-Dorda, S., Wallnau, K., Seacord, R., & Robert, J. (2000, April 1). A Survey of Legacy System Modernization Approaches. (Technical Note CMU/SEI-2000-TN-003). Retrieved September 29, 2023, from https://insights.sei.cmu.edu/library/a-survey-of-legacy-system-modernization-approaches/.

J. Kazanavičius and D. Mažeika, "Migrating Legacy Software to Microservices Architecture," *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream),* Vilnius, Lithuania, 2019, pp. 1-5, doi: 10.1109/eStream.2019.8732170.

T. C. Fanelli, S. C. Simons and S. Banerjee, "A Systematic Framework for Modernizing Legacy Application Systems," *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Osaka, Japan, 2016, pp. 678-682, doi: 10.1109/SANER.2016.40.