From Flex to Angular: The Reincarnation of an Obsolete Web Application

(Technical Paper)

A Case Study of Adobe Flex: The Socio-technical Interactions That Determine the Lifecycle of a Web Framework

(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

Michael Starego

Fall, 2021

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

Advisors

Rider Foley, Department of Engineering and Society

Daniel Graham, Department of Computer Science

Introduction

Software applications do not possess the gift of eternal life. Each one relies on a complex infrastructure that is rapidly evolving due to scientific advancement in the field of computing coupled with the financial incentive for providers of this infrastructure to constantly innovate (Sandborn, 2007). When a software application becomes obsolete, it can have broad impacts on other systems that rely on its functionality. This problem is of particular concern in web application development due to high levels of dependency on external infrastructure. A small web application may depend on the support of multiple web browsers, dozens of third-party libraries, advanced hardware, and specifications of markup languages such as HTML and CSS (Deremuk, 2021). If any of these foundational elements become obsolete, a web application that relies on them may need at best minor refactoring and at worst a complete redesign.

A piece of software is considered obsolete if it is no longer able to function as it was originally intended (functional obsolescence) or if the sale or support for that software has terminated (technological obsolescence) (Bradley & Dawson, 1998). In the web application space, there are many components that can experience each of these types of obsolescence. Functional obsolescence occurs nearly every time a browser is updated: support for certain features is dropped, and third-party libraries and plugins that rely on those features become obsolete. Technological obsolescence can by experienced by reaching a scheduled end of life date. The absence of active support for a piece of software will inevitably lead to functional obsolescence (Bradley & Dawson, 1998).

The problem of web applications depending on obsolete infrastructure is closely related to the work that I do for my internship at the SAS Institute. At SAS, I work as part of a team tasked with managing the company's internal data. This often involves designing database

systems and building web applications for administrators to use. When the internship began in summer 2020, my team was tasked with rebuilding multiple web applications from scratch because the original versions relied on technology that is no longer supported by contemporary web browsers. This internship has demonstrated firsthand the cost of depending on functionally obsolete software. Informed by this experience, I intend to explore the socio-technical interactions that determine the emergence and obsolescence of web frameworks.

Reincarnating an Obsolete Application

My technical report will focus on building one application that was part of the broader effort to replace legacy systems at SAS. The application, dubbed the Software Fee Calculator (SFC), is used by SAS employees to calculate fees for custom software packages and provide quotes to potential customers. The app was originally constructed a decade ago using Adobe Flex, an obsolete web application framework. The goal of this project was to rebuild the application from scratch using contemporary web technologies.

A web application framework, or web framework, is a set of tools that make up a reusable design for a web application (Ignacio Fernández-Villamor, Díaz-Casillas, & Iglesias, 2008). This set of tools normally consists of a software library that provides an application programming interface. It may also include a software development kit that contributes advanced tools such as a command-line interface. Essentially, a framework allows developers to avoid re-inventing the wheel for every application. For example, a key requirement of the SFC is to display data in an Excel-like grid in the user's web browser, and Flex provides a component that meets this requirement right out-of-the-box (Kastwar, 2010). However, the use of a framework

makes an application dependent on the continued support of that technology. Unfortunately for the original SFC, Flex had a rotting foundation: Adobe Flash Player.

At the end of 2020, all major web browsers ended support for Flash, citing major security flaws and performance issues ("Adobe Flash Player End of Life," 2021). This rendered the Flex framework functionally obsolete: Flex itself would be available to use, but it would not be able to run in any contemporary web browser. It is worth noting the distinction between Flash Player and Flex: Flash Player is a tool used to generate animated content in a web browser, while Flex is a web framework that uses Flash as its foundation (Kastwar, 2010). Thus, the fate of Flex is permanently intertwined with that of Flash Player. When the discontinuation of Flash was announced, my team raced to replace all of the apps, including the SFC, that depended on it. If these projects were not completed before browser support for Flash ended, the users of the Flex-based apps would be required to access them through a virtual machine, a scenario that my team wished to avoid at all costs.

The web framework Angular was selected to be the basis for the new software. Like Flex, Angular is an open-source platform. This means that the license of the software conforms to a set of standards set by the Open Source Initiative that require, among other things, that it is free to use and communally maintained ("The Open Source Definition | Open Source Initiative," 2007). However, Angular is a JavaScript framework, making it fundamentally different than Flex. Angular relies on JavaScript, HTML, and CSS, which are part of the native functionality of all major web browsers, while Flex depends on Flash Player, which requires a browser plugin (Bodrov-Krukowski, 2018; Kastwar, 2010). Flash Player itself is not open source. This aspect combined with open-source support makes it very unlikely for Angular to fall into technological obsolescence the same way Flex did.

As previously mentioned, a major requirement of the SFC is to display information in a data grid, and the AG-Grid JavaScript library was used to accomplish this. The AG-Grid library provides many open-source features, but more advanced functionality is only available behind a paywall ("Our Mission, Our Principles and Our Team at AG Grid," 2021). This dependence on proprietary technology creates some of the same issues as Flex had with Flash Player, though replacing AG-Grid in the new SFC would be much less costly than replacing the entire application. There is a delicate balance to be struck between depending too much on the values of a third-party codebase versus writing too much custom code.

At the start of 2021, the Angular version of the SFC officially replaced the Flex one. The new app loads significantly faster, is more responsive, and has a cleaner, more intuitive user interface than the previous version according to informal feedback my team has received. While our experience with Angular was overall very positive, we found that the framework had some drawbacks, most notably high levels of dependency on third-party libraries, a steep learning curve, and unnecessary verbosity in the codebase stemming from the vast amount of customization that Angular allows ("The Good and the Bad of Angular Development," 2020). Despite these issues, my team plans on using Angular for the foreseeable future, though we recognize that, like any web framework, it will not be supported forever.

Socio-technical Context of Web Framework Selection

Bearing in mind my experience at SAS, I plan to explore the socio-technical interactions that result in web development frameworks becoming obsolete. To understand the connection between the human, social, and technical aspects of this problem, the user of a web framework must be reconceptualized as more than an atomic web developer. We should instead define each

user as a social actor based on their affiliations, environments, actions, and identities (Lamb & Kling, 2003). Furthermore, developers are not the only actors that may be considered users of a web framework: customers, managers, and teams of developers can be recognized as separate categories of users, as they all have a distinct relationship with the technology (Pano, Graziotin, & Abrahamsson, 2018).

The technical components of the web development system can be separated into two broad categories: hardware and software. Hardware components such as web servers, with the exception of personal computing devices are often invisible to the aforementioned user groups. On the other hand, software components like web browsers are much more noticeable. Customers can ideally choose to view content through their preferred browser and developers must constantly consider browser compatibility when designing an application. Also, other software components indirectly related to a web framework, such as Q&A forums, can play a large role in its usage. The more users a framework has, the more documentation will be available through forum sites like Stack Overflow, which will make the technology more attractive to development teams and managers. In this way, Stack Overflow can provide value to a web framework by widening the avenues of communication between its users (Feldman, 2017).

Taking this into account, I plan to use the theory of technological momentum as the foundation for my analysis. This theory, developed by Thomas Hughes (1987) in "The Evolution of Large Technical Systems," models technological growth and change based on a loosely-defined pattern of evolution. It identifies the phases of invention, development, innovation, transfer and growth, and competition and consolidation. Hughes argues that these phases do not necessarily occur in that order, and may even occur concurrently. Each phase requires specific skills from a system builder to make critical decisions in order to further the development

process. In my analysis, I will consider each of these phases as they relate to the development of web frameworks. I will identify the system builders of the web world, with emphasis on Steve Jobs, as he pioneered web browsing on mobile devices. Jobs is also relevant to the case of Adobe Flash Player because his endorsement of its competitor was largely responsible for shifting public opinion against the technology (Collins, 2016).

While Hughes' (1987) pattern of evolution provides a model for understanding how technological development proceeds over time, he introduces the concept of momentum to explain the forces that guide the process. In the study of motion, momentum is defined as the product of mass and velocity, and in technological change, Hughes asserts that the organizational components of a system are its mass and its growth rate is its velocity. The analogy suggests that a system with high momentum will be greatly resistant to the effects of outside forces. I intend to identify the organizational components that give web frameworks mass and will quantify their growth rate using historical usage data. The concept of momentum provides a dimension to Hughes' model of evolution that makes the framework well-equipped for predicting growth and change in large technological systems.

Research Question and Methods

My research question is as follows: What socio-technical interactions determine the emergence and obsolescence of web frameworks like Adobe Flex? The results of this research could be used to mitigate the effects of software obsolescence on web application systems by providing tools to identify systems that are at risk of becoming obsolete. Software maintenance generally accounts for 80% of the costs of the software development process, so increased efficiency in this area could greatly reduce the overall cost of an application (Zarnekow &

Brenner, 2005). Furthermore, understanding the ways in which new web frameworks emerge and gain momentum could help teams like mine make well-informed decisions when selecting a new framework.

The method I will use is a case analysis of the rise and fall of the Adobe Flex framework. Flex was a popular framework in the mid-2000s that experienced a sharp decline in the early 2010s due diminishing support for Adobe Flash Player, a technology that Flex depends upon (Collins, 2016). Most notably, Apple's refusal to support Flash Player in its mobile devices paved the way for HTML5, a competing technology with a fundamentally different design philosophy, to conquer the browser world (Collins, 2016; Jobs, 2010; Prince, 2013). In this study, the evidence gathered will include usage data, technical documentation, blog posts, and interviews with developers. Additionally, I will use these methods to consider two of the most popular web frameworks in 2021, Angular and JQuery, and will compare and contrast these technologies with Flex ("Stack Overflow Developer Survey 2021," 2021). From this research, I will be able to identify the sociotechnical interactions that determined the fate of Flex, from its rise to its obsoletion.

Conclusion

Every web framework will eventually become obsolete. When this happens, costly maintenance will be required for all applications that depend on that technology. I plan to research the factors that lead to obsolescence by performing a case study of the Adobe Flex framework. I expect to find that a shift in developer's values can cause obsolescence of a framework or, if the framework is proprietary, the loss of profitability of that product. I expect that these results will be extrapolatable to contemporary web frameworks such as Angular and

JQuery. These results can be used to inform future decisions on the selection of web frameworks and help mitigate the cost of dependence on obsolete software.

References

Adobe Flash Player End of Life. (2021). Retrieved November 1, 2021, from https://www.adobe.com/products/flashplayer/end-of-life.html

Bodrov-Krukowski, I. (2018, March 22). Angular Introduction: What It Is, and Why You Should Use It - SitePoint. Retrieved November 1, 2021, from https://www.sitepoint.com/angular-introduction/

Bradley, M., & Dawson, R. J. (1998). An Analysis of Obsolescence Risk in It Systems. In C. Hawkins, M. Ross, & G. Staples (Eds.), *Software Quality Management VI* (pp. 209–217). London: Springer London. https://doi.org/10.1007/978-1-4471-1303-4_19

Collins, K. (2016, December 29). How Adobe Flash fell to the brink of obscurity—And why it's worth saving. Retrieved November 1, 2021, from Quartz website: https://qz.com/863467/how-adobe-flash-once-the-face-of-the-web-fell-to-the-brink-of-obscurity-and-why-its-worth-saving/

Deremuk, I. (2021, April 22). Web Application Architecture: A Guide Through the Intricate Process of Building an App | LITSLINK Blog. Retrieved November 1, 2021, from Litslink website: https://litslink.com/blog/web-application-architecture

Feldman, B. (2017, March 24). The Hidden Power of Stack Overflow. Retrieved November 1, 2021, from Intelligencer website: https://nymag.com/intelligencer/2017/03/the-hidden-power-of-stack-overflow.html

Hughes, T. P., Bijker, W. E., & Pinch, T. J. (1987). The Evolution of Large Technological Systems. In *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology* (pp. 51–82). MIT Press.

Ignacio Fernández-Villamor, J., Díaz-Casillas, L., & Iglesias, C. Á. (2008). A comparison model for agile web frameworks. *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems*, 1–8. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1621087.1621101

Jobs, S. (2010, April). *Thoughts on Flash*. Retrieved from https://newslang.ch/wordpress/wp-content/uploads/2020/06/Thoughts-on-Flash.pdf.

Kastwar, A. (2010, September 30). Adobe Flex Platform: A leading solution to build Rich Internet Applications (RIA). Retrieved November 1, 2021, from https://blog.e-zest.com/adobe-flex-platform-a-leading-solution-to-build-rich-internet-applications-ria/

Lamb, R., & Kling, R. (2003). Reconceptualizing Users as Social Actors in Information Systems Research. *MIS Quarterly*, *27*(2), 197–236. https://doi.org/10.2307/30036529

Our Mission, Our Principles and Our Team at AG Grid. (2021). Retrieved November 1, 2021, from https://www.ag-grid.com/about.php

Pano, A., Graziotin, D., & Abrahamsson, P. (2018). Factors and actors leading to the adoption of a JavaScript framework. *Empirical Software Engineering*, *23*(6), 3503–3534. https://doi.org/10.1007/s10664-018-9613-x

Prince, J. D. (2013). HTML5: Not Just a Substitute for Flash. *Journal of Electronic Resources in Medical Libraries*, *10*(2), 108–112. https://doi.org/10.1080/15424065.2013.792561

Sandborn, P. (2007, December). *Software Obsolescence – Complicating the Part and Technology Obsolescence Management Problem*. IEEE Trans on Components and Packaging Technologies.

Stack Overflow Developer Survey 2021. (2021). Retrieved November 1, 2021, from Stack Overflow website: https://insights.stackoverflow.com/survey/2021/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2021

The Good and the Bad of Angular Development. (2020, March). Retrieved December 6, 2021, from AltexSoft website: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/

The Open Source Definition | Open Source Initiative. (2007, March 22). Retrieved November 1, 2021, from https://opensource.org/osd

Zarnekow, R., & Brenner, W. (2005, January 1). *Distribution of Cost over the Application Lifecycle—A Multi-case Study.* 68–79.