# The Emergence and Obsolescence of Web Frameworks

A Research Paper submitted to the Department of Engineering and Society

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

**Michael Starego**

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

Advisor

S. Travis Elliott, Department of Engineering and Society

## 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. Nearly all modern web applications are built on top of web frameworks, which are sets of software tools that provide reusable high-level application designs. These frameworks may lower development costs, but they often depend on dozens of third-party libraries, support from 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.

Motivated by this problem, this paper will explore the sociotechnical interactions that determine the emergence and obsolescence of web frameworks. This analysis will be accomplished through a case study of Adobe Flex, a web framework popular in the late 2000s that experienced a steep decline in usage and support throughout the 2010s. We will employ Thomas Hughes' theory of technological momentum, which models technological growth and change based on a loosely-defined pattern of evolution, to guide us through the key inflection points in the history of Flex. This analysis will be supplemented by interviews from web developers who have used Flex in a professional environment. The results from this study of Flex will be extrapolated and applied to contemporary technologies, towards identifying the conditions that determine the growth and decline of a web framework.

**What is a Web Framework?**

Web frameworks like Adobe Flex typically provide a set of software tools, which may include a collection of pre-designed user interface components, a software library to provide common web-specific functionality (such as making HTTP requests), a command line interface, a compiler, or some combination of these (Ignacio Fernández-Villamor, Díaz-Casillas, & Iglesias, 2008). Broadly speaking, the goal of these tools is to provide scaffolding for a web application so that the developer doesn't have to "reinvent the wheel" for every new project. This can greatly reduce development time and make web applications much easier to maintain. If a developer has experience working with a given framework, they should be able to quickly understand the code base of other apps using the same design.

Due to these advantages, the vast majority of web applications today are built using a framework. This gives a great deal of influence to the individual developers and organizations that produce these ubiquitous software tool sets. In 2021, two of the most popular web frameworks were React, which is maintained by Meta (formerly Facebook), and Angular, maintained by Google ("Stack Overflow Developer Survey 2021," 2021). In both cases, the software is open-source and provided free of charge. These corporations wield the power of the open-source development community to report and fix bugs, provide feedback on the design, and suggest new features. This process results in a high-quality code base, which is of great benefit to the corporation that maintains it as well as the web development community as a whole.

Like React and Angular, Adobe Flex is an open-source platform, with one major caveat: while Flex itself is free to use and communally maintained, it is heavily dependent on Adobe Flash Player, which itself is proprietary (Kastwar, 2010). Without the Flash Player plugin installed, a user cannot run a Flex app in their browser. In contrast, React and Angular employ

the browser's native functionality: HTML, JavaScript, and CSS (Bodrov-Krukowski, 2018). While this may seem to be an obvious design flaw of Flex, in the late 2000s, when Flash Player was installed on nearly every internet-connected personal computer, its complete obsoletion merely a decade later would likely have seemed unfathomable.

**Application of Technological Momentum**

This paper will explore the history of Adobe Flex according to the pattern of growth and change proposed by Thomas Hughes in "The Evolution of Large Technological Systems." In this essay, Hughes (1987) identifies the phases of invention, development, innovation, transfer, growth, competition, and consolidation. During invention, a radical new idea is produced. In development, that idea is turned into a technology. During innovation, that technology is brought to the market. During transfer, that technology is modified by other developers. In growth, competition, and consolidation, multiple variations of the technology arise and compete until one is ultimately accepted. Hughes argues that these phases do not necessarily occur in that order, and may even occur concurrently. Each phase requires specific skills from an individual that Hughes calls a system builder to make critical decisions in order to further the development process. Thus, the history of a technological system can be interpreted by identifying each of these phases of development and the system builders that made them possible.

While this provides a model for understanding how technological development proceeds over time, Hughes (1987) introduces an additional concept, which he terms 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. This analogy suggests that a system with high momentum will be greatly resistant to the effects of outside forces. Hughes

uses this concept to explain why large, mature technological systems appear to exhibit a form of determinism. He argues that the system is itself not autonomous, but has accrued enough momentum that its continued forward progress seems inevitable.

Extending Hughes' (1987) analogy to the web application system, we find that the infrastructure that supports a web framework to be its mass and its number of users its velocity. A framework's infrastructure consists of a vast network of software and hardware components, including web browsers, online documentation, data transfer protocols, wireless networks, database servers, etc. Additionally, the user of a web framework should be conceptualized as more than an individual web developer (Lamb & Kling, 2003). In this paper, we will instead define each user based on their affiliations, environments, actions, and identities. 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). Put simply, the user of a web framework is any actor that has a social stake in its support and usage.

**History of Adobe Flex**

We will now examine the history of Adobe Flex according to Hughes' pattern of evolution. Because Flex itself is not a technological system but an example of one technology within a system, we will consider the phases of evolution of the web development system as a whole, while focusing on the technologies that allowed for rich internet applications (RIAs) – web applications that provide an interactive user experience traditionally associated with desktop applications. RIAs can be viewed in contrast to traditional page-based web apps, which are static text documents that only allow user interaction through form elements such as inputs and check boxes. RIAs can exhibit a wide range of interactive features, from dragging and dropping

elements on a page to fully-functional graphics-based video games in the browser (Kay, 2009). This distinction is made because Flex is a framework for building RIAs, so narrowing the scope of this investigation in this way will provide more insights into the growth and decline of Flex.

*Invention and Development*

The history of rich internet applications began with the introduction of the Java programming language in 1995. With the internet in its infancy, the creators of Java sought to capitalize on the need for more interactive content on the web and in 1997 released HotJava, a web browser that could run Java applets – small Java applications capable of producing interactive user interfaces (Topic, 2016). Unlike most web content of the time, which was delivered to the user in the form of static HTML documents, applets were sent as Java bytecode (Java code compiled into binary machine code) and executed directly on the user's machine (Topic, 2016). This was a revolutionary step forward for the web because browsers of the time were largely incapable of executing code natively. However, this approach had some major downsides. Java bytecode files were often large and took longer to transmit than standard text-based HTML, and programming Java applets was laborious and offered a limited set of features (Evans, 2015, p. 10; Topic, 2016). Despite these issues, Java applets provided a first glimpse into the world of RIAs and paved the way for the development of two platforms that would usher in the next generation of web content: AJAX and Adobe Flash Player.

In early 2005, Jesse James Garett (2005) published "Ajax: A New Approach to Web Applications" and took the web world by storm. This paper proposed a way to provide interactive content on the web without having the user reload the page. This meant that once a web application was loaded, interactions with the web server could happen seamlessly in the background, creating a truly dynamic user experience on the web. Arguably the most significant

aspect of Garett's proposal was that it required only existing technologies native to all popular browsers of the time: markup languages HTML and CSS as well as scripting language JavaScript. Unlike Java applets, these languages are delivered as text and interpreted by the user's browser. This aspect hinted at a future in which browsers didn't need to rely on third-party plugins to run rich applications which could eventually replace desktop apps. Garett, a system builder, saw this radical invention as the future of the web development world, and worked to see his vision to fulfillment.

Later that year, software juggernaut Adobe acquired Macromedia, Inc and in doing so inherited one of the most significant programs in the history of the web: Flash Player (McElhearn, 2020). At its core, Flash Player was an animation software that provided rich content, video, and audio in a web browser (Collins, 2016). Similarly to applets, Flash users were required to install a third-party plugin in order to view the content. This design is in contrast to that of AJAX, which relies exclusively on native browser functionality. Flash's approach offered a significant benefit at the time: universality. Developers could rely on Flash content to appear the same in any browser it was executed in, while HTML, CSS, and JavaScript semantics differed slightly between browsers, often with frustrating results (McElhearn, 2020). Additionally, Flash supported video and audio content, which was not possible through native browser technologies (McElhearn, 2020). This set the stage for a competition between two design philosophies that would last through the end of the decade: native browser features vs Flash Player.

*Innovation and Transfer*

About a year after Garett's paper was published, another system builder, a software developer named John Resig, published a JavaScript library called jQuery that realized Garett's

revolutionary design (Resig, 2006). Using the AJAX design, the jQuery framework allowed web applications to make requests to web servers in the background, without interrupting the user. In addition to this, jQuery provided many useful features for dynamically updating the content and styling of the page. This formed a nearly-complete package for web developers to create rich applications. By the year 2015, jQuery would be used by the majority of the web's top 1 million websites ("JQuery Usage Statistics," n.d.).

The only features jQuery did not provide at the time were the ones that Flash arguably did best: audio and video. Thus, Flash remained widely used during the late-2000s despite jQuery's ubiquity (McElhearn, 2020). To increase their market share, Adobe encouraged web developers to use the Flash platform by providing a web framework of their own: Adobe Flex. Flex harnessed the animation power of Flash Player to provide a large library of pre-designed user interface components that made development simple and accessible (Kastwar, 2010). In 2010, Adobe increased the incentive for developers to use Flex by introducing Flash Builder, a development environment for building Flex apps. Flash Builder featured a user interface through which a developer could design the layout of a page visually, which was then converted to code automatically. This tool became very popular amongst Flex developers. In contrast to Flex itself, which was open-source, Flash Builder was proprietary and sold directly by Adobe. Just as jQuery realized the potential of AJAX, Flex, accompanied by Flash Builder created an efficient, intuitive system by which developers could build rich web applications on top of the Flash Player platform.

*Growth, Competition, and Consolidation*

During the mid-2000s, AJAX and Flash were in a dead heat to determine the future of rich web content. AJAX applications built with jQuery made full use of native browser features.

Flash applications built with Flex behaved more consistently between different browsers and offered the powerful development tools of Flash Builder. However, this competition would eventually be settled by the strong opinions of a powerful system builder: Apple CEO Steve Jobs.

In 2007, Jobs released the iPhone, a revolutionary product that allowed users to browse the internet via a pocket-sized device. Unfortunately for Adobe, the iPhone shipped with Apple's proprietary web browser, Safari, which did not support Flash content (McElhearn, 2020). This was due to Jobs' strong objections to Flash as a technology (Jobs, 2010). He defended his decision to not support Flash in a 2010 open letter titled "Thoughts On Flash," in which he argued that Flash was inefficient, had major security flaws, and had poor support for small screen sizes. He cited the newest update to the HTML standards, termed HTML5, as a complete replacement for Flash as it allowed browsers to natively support audio and video. Though most of the video content on the web still required Flash at the time, Jobs argued that this would soon be replaced by HTML5 as the industry moved away from Flash.

This letter provided a critical blow to Flash at a time when HTML5, its competitor, was rapidly growing in popularity (McElhearn, 2020). In the early 2010s, new web frameworks AngualrJS, React, and Vue were released. Powered by HTML5, these frameworks provided more advanced functionality out-of-the-box than jQuery. Furthermore, in 2011 Google released V8, a new JavaScript interpreter that was soon adopted by many contemporary browsers. V8 allowed browsers to run JavaScript, an interpreted language, with efficiency near that of compiled languages like Java and C++ (DiPierro, 2018; Gibb, 2019). This result was revolutionary and allowed for complex JavaScript programs to run inside web pages, further expanding the possibilities for rich applications using native browser features.

By the late 2010s, Jobs' prediction had come true. Very little, if any, new websites were being developed using Flex or Flash Player. Flash Player's security issues had become more critical, and Adobe had to choose between investing a great deal of resources into re-designing the platform or to abandon it entirely (McElhearn, 2020). Ultimately, they chose the latter. In 2017, Adobe announced that they would cease support for Flash Player at the end of 2020 ("Adobe Flash Player End of Life," 2021). At that point, browsers would no longer allow the Flash Player plugin to run, making all Flash-based web content virtually unusable. In light of this, Adobe donated Flex to the Apache Foundation, where it resides as of 2022. The framework has been renamed to Apache Flex, and still supports generating Flash Player content for mobile and desktop platforms. Apache also provides a tool named Apache Royale that aids in converting Flex apps to JavaScript. While the continued support of Flex is useful for handling legacy applications that depend on Flash Player, no web development is currently being done using the Flex or Flash Player platforms.

**Interviews With Flex Developers**

As part of this study, one-on-one interviews were conducted with three developers that have used Flex in a professional environment. The interviewees, all employees of the same organization, developed separate internal-use web applications using Flex between the years 2010 and 2013. These interviews provide insight into the reasons that Flex was selected by developers, the overall usability of the framework, and how existing applications were handled once support for Flash Player ended.

When asked what web frameworks (if any) were being used before Flex, the interviewees cited Java applets, Struts (a Java-based framework popular in the early 2000s) and Dojo (a pre-jQuery AJAX-based JavaScript framework). While most of the developers maintained some

positive views of these early technologies, they agreed that the switch to Flex was a welcome one. Flex was far easier to work with than Java-based frameworks and there were fewer issues with browser compatibility as there was with JavaScript-based frameworks. In addition to these benefits, one interviewee cited professional development as a reason to start working with Flex. It was a popular technology at the time, and was generally seen as a valuable skill to have. Another developer decided to use Flex on their project because other teams in the company already supported it. A set of user interface components had already been built with the company's approved style and branding, and this greatly reduced the barrier of new teams adopting the technology. This is evidence that there are organizational factors beyond a developer's individual preferences that inform the selection of a web framework.

Two of the three developers characterized their experience with Flex as positive overall. They described the framework as powerful and easy to learn, particularly through the use of Flash Builder. The third developer had a negative view of using Flex, citing difficulty with customizing the framework's built-in features for the application's specific requirements. It is worth noting that this developer was the only one that developed in Flex without using Flash Builder. Additionally, the interviewees cited the documentation and forum support as a strong point of Flex. The depth of the forum support, an important aspect of a framework's infrastructure, is evidence that it was widely used at the time.

When developing their Flex apps, two of the three developers knew that the industry was moving away from Flash Player, though they didn't foresee that support for Flash would end so suddenly. These developers stated that Flex was used despite these concerns because there wasn't an alternative that had been fully vetted by the company at the time. In the years following, the company would switch to Angular, a JavaScript-based framework. By then, cross-

browser compatibility for JavaScript applications was far less of a concern than it was in the early 2010s due to the maturity of modern browsers. Also, the inter-company infrastructure that made Flex originally so appealing to development teams was eventually reconstructed for Angular.

In 2016, when the Flash Player end-of-life was announced, significant effort was made to convert all Flex apps to Angular. As of mid-2022, all new internal-facing web development at the company is done using Angular. Despite this, multiple interviewees believe that Flex has some advantages over Angular, and that it would be a viable option today if Flash Player were still supported. In particular, they noted Angular's steep learning curve compared to Flex. Also, they said that positioning elements on the screen was much easier in Flex than it is in Angular, citing frustration with CSS styling, concluding that this difficulty can lead to increased development time. Finally, when asked whether they think JavaScript frameworks like Angular will ever become obsolete as suddenly as Flex did, the consensus was a firm "no." They concluded that the infrastructure surrounding JavaScript development is so vast that its obsoletion would have to happen very slowly over many years, if at all. They believe that Angular is here to stay for the foreseeable future, but are open to the idea of adopting a new framework that improves on Angular's weak points, provided that the infrastructure is there to support it.

**Discussion**

The case of Adobe Flex provides a unique example of a web framework's infrastructure crumbling over a shockingly short period of time. Consistent with Hughes' (1987) analogy of technological momentum, when the central piece of Flex's infrastructure, Flash Player, was threatened, the community support for the framework went into decline and eventually fell to

zero. As evidenced by the developer interviews, companies may build their own infrastructure around a web framework, but are quick to change course if they sense that the framework may not be viable in the long-term. Ultimately, the negative outlook for Flex was caused by lack of Flash Player support in mobile devices, led by the vocal opinion of Steve Jobs, who argued that HTML5 was simply a more versatile technology and that Flash was never designed to be used on small screens (Jobs, 2010). Thus, the force that was strong enough to buckle the Flash infrastructure and the great amount of momentum that it had accrued was a seismic technological shift in the industry: the advent of mobile internet browsing. If it hadn't been for the iPhone, Steve Jobs' opinions would very likely not have had nearly enough influence to bring down Flash.

In the aftermath of the HTML5 vs Flash Player competition, the frameworks that arose seemed to build momentum in the same way that Flex did. These frameworks, nearly all of which were JavaScript-based, promised unique technological benefits over their predecessors and some (particularly React and Angular) were backed by large organizations. They gradually grew their number of users and with that came increased forum support and improvements on their open-source code bases.

While these modern frameworks compete with one another to some extent, there is little incentive for the market to consolidate around one of them because they are free to use and based on the same underlying technologies. The internet infrastructure is growing so rapidly that acquiring momentum is not a zero-sum game. Many web developers are well-versed in multiple frameworks in order to make their services marketable. This returns to the point that users of a web framework are not just individual web developers but anyone with a stake in the technology, and it is very possible for an actor to have a vested interest in more than one of these

frameworks. Due to the large number of frameworks available to developers today, many of which have a great deal of momentum, it would take another groundbreaking technological shift in the web industry to change the fact that JavaScript frameworks are here to stay.

## Conclusion

In summary, the state of the infrastructure underlying a web framework often determines its emergence and obsolescence. As is demonstrated by the case of Adobe Flex, when a central element of a framework's infrastructure is removed, a technology can very quickly lose momentum and become obsolete. Interviews with web developers that have worked with Flex revealed that this infrastructure has components that go beyond browser compatibility, namely support from within an organization and documentation created through online forums. In light of these results, we can conclude that the current state of the web industry, which is dominated by JavaScript frameworks with great amounts of momentum, will be resistant to rapid change even as new technologies arise.

**References**


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/

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

DiPierro, M. (2018). The Rise of JavaScript. *IEEE Computer Society*, *20*, 9–10. https://doi.ieeecomputersociety.org/10.1109/MCSE.2018.011111120

Evans, B. (2015). *Java: The Legend*. O'Reilly Media, Inc.

Garett, J. J. (2005, February 28). *Ajax: A New Approach to Web Applications*. Adaptive Path. Retrieved from https://immagic.com/eLibrary/ARCHIVES/GENERAL/ADTVPATH/A050218G.pdf

Gibb, R. (2019, September 17). What is V8 JavaScript Engine? Retrieved March 20, 2022, from StackPath website: https://blog.stackpath.com/v8-javascript-engine/

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.

JQuery Usage Statistics. (n.d.). Retrieved March 20, 2022, from BuiltWith website: https://trends.builtwith.com/javascript/jQuery

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/

Kay, R. (2009, April 6). Rich Internet Applications. Retrieved March 20, 2022, from Computerworld website: https://www.computerworld.com/article/2551058/rich-internet-applications.html

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

McElhearn, K. (2020, December 29). The History of Adobe Flash Player: From Multimedia to Malware. Retrieved March 20, 2022, from The Mac Security Blog website: https://www.intego.com/mac-security-blog/the-history-of-adobe-flash-player-from-multimedia-to-malware/

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

Resig, J. (2006, January 16). BarCampNYC Wrap-up. Retrieved March 20, 2022, from John Resig website: https://johnresig.com/blog/barcampnyc-wrap-up/

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

Topic, D. (2016, January). *Migrating from Java Applets to plugin-free Java technologies*. Oracle Corporation. Retrieved from https://www.oracle.com/technetwork/java/javase/migratingfromapplets-2872444.pdf