Javier A. Arroyo-Solis, Nasra Muhumed, and Jenna Suits

Dr. Christopher Harris

CS 395-001 Emerging Topics in Computer Science

February 25, 2024

Technical Writing Essay - TypeScript

Abstract

Java, JavaScript, and TypeScript are key programming languages in web development and software engineering, each with distinct characteristics. This paper examines why TypeScript may be advantageous over Java and JavaScript in certain scenarios. Despite the established dominance of Java and JavaScript, TypeScript offers unique features that enhance developer productivity, code maintainability, and scalability. Our analysis suggests that TypeScript's static typing, advanced features, and expanding ecosystem make it a compelling alternative. By leveraging TypeScript's capabilities, developers can detect errors early, write more expressive code, and access a growing set of frameworks and tools. We explore case studies of developers transitioning from TypeScript, underlining the importance of user choice and diversity in software development. Ultimately, TypeScript emerges as a valuable option, benefiting developers and projects across various contexts.

Introduction

Java, JavaScript, and TypeScript are three prominent programming languages in the realm of web development and software engineering. Each of these languages serves distinct purposes and has its own set of features, advantages, and challenges. In this paper, our objective is to explore why TypeScript might be preferable compared to Java and JavaScript in certain contexts. While Java and JavaScript have established themselves as dominant players in the programming landscape, each with its own strengths and areas of application, TypeScript offers unique features and benefits that make it an attractive choice for modern software development. Our hypothesis is that TypeScript's combination of static typing, advanced features, and growing ecosystem positions it as a compelling alternative to both Java and JavaScript in scenarios where developer productivity, code maintainability, and scalability are paramount. By leveraging TypeScript's static typing, developers can catch errors at compile-time, thereby enhancing code

reliability and reducing the likelihood of runtime failures compared to dynamically typed languages like JavaScript. Additionally, TypeScript's support for advanced language features such as interfaces, generics, and decorators empowers developers to write more expressive, modular, and scalable code, leading to increased productivity and code maintainability over traditional JavaScript.

“Java is a widely used programming language because it is well known for its performance, platform independence, and security.”(Martinez et al. 2). Java emerged in the mid-1990s as a response to the complexities and limitations of existing programming languages. Developed by a team led by James Gosling at Sun Microsystems, Java was designed to be platform-independent, secure, and robust. Its syntax drew inspiration from C and C++, making it familiar to a wide range of developers. Java was meticulously crafted to be platform-independent, a feat achieved through the innovative concept of the Java Virtual Machine (JVM). “Java was designed with a Java compiler that converts the source code to intermediate code called byte code. This code is then run by a Java Virtual Machine because it is not machine specific.”(Developer). This breakthrough allowed Java programs to run on any device equipped with a compatible JVM, regardless of the underlying hardware or operating system, paving the way for unparalleled portability and scalability.

In addition to its widespread adoption and enduring legacy, Java continues to evolve in response to emerging trends and technological advancements. With the rise of cloud computing, Java remains a preferred choice for developing scalable and resilient cloud-native applications. The Java community actively contributes to open-source projects and standards bodies, driving innovation and ensuring Java's relevance in an ever-changing landscape. “Java is a general-purpose programming language designed to enable programmers to write code that is universally compatible and can be executed on multiple platforms without requiring any recompilation.”(Martinez et al. 2). Moreover, Java's versatility extends beyond traditional software development, with increasing applications in areas such as data science, artificial intelligence, and edge computing. As Java celebrates decades of success, its commitment to performance, platform independence, and security remains unwavering, solidifying its position as a cornerstone of the digital age and paving the way for a future filled with endless possibilities.

Often referred to as the "language of the web," JavaScript was created in the early 1990s by Brendan Eich, an engineer at Netscape Communications Corporation. With the official release of JavaScript and Netscape Navigator 2.0 in 1995, dynamic behavior was added to web pages, completely changing how content was shown and interacted with on the internet. “Live Script was the first name of JavaScript and made its first appearance in Netscape 2.0. Later on, Live Script was changed with Java Script by Netscape Communications Corporation. In 1997, JavaScript 1.1 was submitted to the ECMA (European Computer Manufacturers Association). The ECMA–262 specifications defined a standard version of the core JavaScript language.” (Zobair Ullah, 37137). In contrast to its forerunners, which were mostly used as markup languages to organize web content, JavaScript elevated responsiveness and interactivity to the fore, allowing programmers to create dynamic websites. JavaScript gained popularity due to its ease of use and adaptability, and its acceptance throughout the expanding World Wide Web spread quickly.

Since its launch, JavaScript's impact on web development has only increased. It is still one of the key technologies used today to create dynamic and interesting online apps. Because of its adaptability, programmers can design a wide range of functionalities, from straightforward animations to intricate real-time systems. JavaScript has made it possible to create single-page applications (SPAs), which offer a seamless user experience by dynamically updating content without requiring page reloads. Examples of these frameworks and libraries are React, Angular, and Vue.js. In the future, JavaScript will continue to evolve due to continuing standardization initiatives like ECMAScript updates, which will provide new functionality and enhancements to the language. Furthermore, new tools, libraries, and frameworks are continuously being produced in the JavaScript ecosystem to meet the changing needs of web development.

TypeScript, a powerful superset of JavaScript that was created to overcome some of the language's shortcomings and improve developer productivity and code maintainability, has grown in importance by building on the popularity and success of JavaScript. Developed by Microsoft, TypeScript was first unveiled to the public in 2012. Leading this effort was Anders Hejlsberg, a seasoned language designer well-known for his contributions to important computer

languages like Delphi, C#, and Turbo Pascal. Hejlsberg and his Microsoft colleagues saw that a language was needed to close the gap between the flexibility of JavaScript and the requirements of contemporary software engineering methods. “Despite its success, JavaScript remains a poor language for developing and maintaining large applications. TypeScript is an extension of JavaScript in- tended to address this deficiency.”(Bierman et al. 1). TypeScript introduces static typing, allowing developers to catch errors at compile-time rather than runtime, thus improving code reliability. Additionally, it offers advanced features such as interfaces, generics, and decorators, empowering developers to build scalable and robust applications with ease. This combination of features has made TypeScript increasingly popular among developers and has led to its adoption in a wide range of projects, from small-scale applications to large enterprise systems.

Beyond its technical merits, TypeScript has a vibrant and rapidly growing community of developers, contributors, and enthusiasts. This community actively contributes to the language's evolution through feedback, bug reports, and open-source contributions. Furthermore, TypeScript's collaborative development process and open architecture encourage creativity and experimentation, which has resulted in the creation of a large number of libraries, frameworks, and tools made especially for TypeScript development. As TypeScript's ecosystem grows, developers have access to a multitude of tools that can expedite their projects, ranging from well-known frontend frameworks like Angular and React to backend frameworks like NestJS. Furthermore, TypeScript is kept at the forefront of contemporary software development methods by the Microsoft TypeScript team through frequent updates and releases. As a result, TypeScript continues to gain traction not only within the JavaScript ecosystem but also in diverse fields such as cloud computing, machine learning, and Internet of Things (IoT) applications.

Comparative Analysis

In continuation, TypeScript has many similarities to JavaScript and Java which have been released before TypeScript. To begin, TypeScript can simply be described as JavaScript with more features. This means that every JavaScript code can be used successfully in TypeScript. Although there are many similarities there are many differences as well. The learning curve between the two coding languages is just one difference. For TypeScript you need to understand

JavaScript because of TypeScript being a superset of it. You also need to have an understanding of OOPS, object-oriented programming system, as well. Now on the other side, JavaScript is well known and has an “easy-to-learn scripting language. Many developers use JavaScript with CSS and HTML to create web applications” (Raval,10). Although JavaScript is much easier to learn and use, TypeScript has many benefits if used, that makes it worth it in the end.

The next difference between JavaScript and TypeScript is the syntax of both programming languages. Looking at TypeScript, it consists of, “variable declaration, functional paradigm, and type syntax, which JavaScript doesn’t offer” (Raval, 10). TypeScript is considered to be a more modern language . While TypeScript is a typed language, JavaScript is a scripting language, a combination of other languages’ structured programming terminologies. Although TypeScript changes slightly how variables are referenced and declared, it helps drastically on the code’s consistency. This is because it sets the variable type so that it cannot be arbitrarily changed later within the code. Because TypeScript is organized in a better way than JavaScript, it allows it to be more capable of large-scale applications. According to Cameron McKenzie, “With TypeScript, when a variable is assigned a new value, IDEs and validation tools can quickly check that change against tens of thousands of lines of code. The same type of check is difficult to perform with JavaScript. This increases the risk of error when extending, updating and troubleshooting JavaScript” (McKenzie, 2). TypeScript in turn has better error handling because TypeScript’s code has static type checking. This means that it catches bugs before running which results in less runtime errors within the code. Overall TypeScript identifies 15 percent of JavaScript errors. Looking back, TypeScript changes small things, compared to JavaScript code, and has immense benefits like more organization and more errors found before running, which results in less time trying to debug your code.

Continuing, TypeScript consists of many frameworks that have gained major popularity. The first one is NestJs. This framework is the most popular among TypeScript frameworks. According to Soloman, “NestJs is one of the fastest-growing frameworks in the NodeJs ecosystem. It has outgrown many other NodeJs frameworks” (Soloman, 3). NestJs consists of 99.8 percent TypeScript in the codebase. NestJs is considered progressive for building NodeJs

applications with TypeScript that are more efficient and scalable. NestJs allows for the making of applications that are easily testable and scalable.

Another framework is FeatherJs. FeatherJs's codebase consists of 92.1 percent TypeScript. This makes this framework TypeScript's second most popular framework. FeathersJs is considered a framework for creating API's and real-time applications. FeathersJs is very helpful when it comes to being compatible with many different technologies like Android and iOS. FeatherJs is considered to be simple and flexible. FeatherJs is a service-oriented thinking model, which is regulated and maintained by the community, FeatherJs is engine-agnostic which in turn means that it can run on client as well as server sides.

The next framework is LoopbackJs. LoopbackJs is considered to be highly scalable in TypeScript and is known for building microservices and APIs. Looking at its latest update, Loopback 4, it has officially been changed to fully TypeScript. Originally LoopbackJs's codebase consisted of 73.9 percent of TypeScript. LoopbackJs is based on Express and gives you the option to create APIs quickly. LoopbackJs is known for being able to be small, fast and flexible, making it easy for small and large teams when creating a new application. Although it could be considered a REST framework it offers other aspects as well for more flexibility and more complex projects.

Continuing in frameworks, AdonisJs is next. AdonisJs is considered to be easy to learn if you have prior knowledge of Laravel, Spring, or even Ruby on Rails. AdonisJs's codebase consists of 99.9 percent of TypeScript. According to MasteringBackend, "AdonisJS is a Node.js framework focused on developers' ergonomics, stability, and speed. AdonisJS is written from the ground up with a strong principle and goals in mind to be a strong integrated system." (MasteringBackend, 1). It is considered to be focused on developer experience, scalability, as well as speed.

The last framework is Ts.Ed. Ts.Ed is written with TypeScript as well as ExpressJs and Koa, this helps to build the server application in a fast manner. Ts.Ed's codebase consists of 98.4

percent of typescript. Ts.Ed is known for fast preconfiguration and creation of easy rest API. Ts.Ed uses object oriented programming along with functional programming.

Continuing, Typescript has become a major influence on startups. The first startup is Slack, which is a business communication platform. Along with Slack, there have been many other startups like Kavak, Bitpanda, DoorDash, even Canva. There are many reasons as to why more developers are using TypeScript for their startups and that is because it is the only language to build an application that is frontend, backend, and mobile. According to Andrew Lee, "TypeScript provides just enough structure to satisfy static type advocates and just enough flexibility for dynamic type advocates. It's the happy medium that will prevent your team from looking for something else" (Lee, 4). Another reason many others are using TypeScript for startups is because it helps to detect 90 percent of bugs during the development of the application. According to Harish Neel, TypeScript allows for enough structure that static type programmers like along with enough flexibility for those programmers that like dynamic type programming. Overall TypeScript offers many benefits to startups and although it may take more time to learn and use the language the benefits outweigh this, with the debugging and static typing this programming language has many appeals.

Discussion

Over the years with TypeScript, the idea of TypeScript being a better option compared to that of JavaScript has been becoming more and more popular. For many programmers and developers, the additions that TypeScript provides to the user has made the switch from base/vanilla JavaScript to TypeScript a no-brainer to many users even with the drawback of needing to learn the necessary information in order to use TypeScript. But even with this common opinion in place, some big open-source projects have felt it would be more beneficial to move away from TypeScript and move back to plain JavaScript. Two examples of this are David Heinemeier Hansson with his project Turbo 8 and Rich Harris with Svelte.

David Heinemeier Hansson, the creator of Ruby on Rails, has been creating Turbo and its newest version Turbo V8 which is mainly used as a library for Ruby on Rails pages. In a blog post, he stated he is moving away from TypeScript for the Turbo V8 project with the reasoning

being that as he stated “but because it pollutes the code with type gymnastics that add ever so little joy to my development experience, and quite frequently considerable grief. Things that should be easy become hard, and things that are hard become `any`.” (Hansson). He would later in the blog post state that “While you may compile dialects into it, you still have to accept the fact that running code in the browser means running JavaScript. So being able to write that, free of any tooling, and free of any strong typing, is a blessing under the circumstances.” (Hansson). From this, we can see that Hansson views the addition of type-casting, one of the main benefits of TypeScript, as a hindrance for him and his large library he is creating as coding in the right typing without errors can be difficult and sometimes results in him setting the type to ‘any’ just avoid the trouble. Being able to avoid this problem all together with the use of plain JavaScript is to him more benefit.

Next is Rich Harris, who is the creator of Svelte, an open-source component-based front-end software framework used to build UI components, and is also moving away from TypeScript back to plain JavaScript but not for the same reason as before. The main reason for this move is to avoid the compile step that is necessary for TypeScript files. Since TypeScript is a superset of JavaScript, browsers can not interpret the TypeScript code and needs to transpile it back to JavaScript in order for the browsers to run it. This can decrease productivity especially for larger frameworks such as Svelte but now how do they manage the code? Well from the post from Rich Harris, the solution for this is to use an API which uses JavaScript own documentation comments to generate types similar to that of TypeScript without the need of the extra compile step. With the use of this tool, Svelte developers can retain the benefits of TypeScript such as type-casting and early bug detection without the need of the compile step which increases the productivity for the team there.

A likely question being asked is what’s the reason for showing these cases of developers moving away from TypeScript. The answer for this would be to show not only that TypeScript is not a perfect language but also to show that it's up to the user to choose what would be best at a given situation. While it is true that many view TypeScript as the better option, it’s important that there are different options to users depending on the needs and what they are trying to do at a given moment. It is also important to know that these cases that were shown came up in recent

years so well this opinion of TypeScript may continue to rise, it will likely stay overshadowed by the more popular opinion in the tech community. But in these cases, we see developers pick and choose different pathways which depending on their choice would provide better productivity, better code maintainability, and better scalability in their eyes. Maybe it’s better to move away from the over-complicated and not feel shackled to that of type-casting, or maybe it’s better to find alternative tools to use instead, or maybe it’s better to weigh the positives with the negatives and see if one outweighs the other. It comes down to user choice, which for many people still is TypeScript even with the negatives as for many the positives outweigh them or to some the negatives are negligible.

Conclusion

Returning to our hypothesis that TypeScript's combination of static typing, advanced features, and growing ecosystem positions it as a compelling alternative which helps developers with productivity, code maintainability, and scalability, this essay shows that this hypothesis to be true. With static typing, users are able to catch errors before runtime, increasing the productivity but also assist with code readability which helps with both code maintainability as well as scaling the project to a larger size. The advanced features of TypeScript helps to streamline the experience well using TypeScript, thus increasing the productivity and code maintainability over all. Lastly, with the increasing popularity of TypeScript, the growing ecosystem of it from not just the main developers from Microsoft but also from contributors and enthusiasts of the language. This will only continue as TypeScript remains relevant resulting in more productivity for the users and a help on the scalability of different projects that use TypeScript.

Citations

contributors, daffl, marshallswain, and FeathersJS. “Feathers.” *Feathersjs.com*, 15 Feb. 2024, feathersjs.com/feathers-vs-loopback.html. Accessed 24 Feb. 2024.

“@Feathersjs/Feathers.” *Npm*, 15 Feb. 2024, www.npmjs.com/package/@feathersjs/feathers. Accessed 24 Feb. 2024.

Eseme, Solomon. “AdonisJS Tutorial: The Ultimate Guide (2023).” *Mastering Backend*, 8 Feb. 2021, masteringbackend.com/posts/adonisjs-tutorial-the-ultimate-guide. Accessed 24 Feb. 2024.

Eseme, Solomon. “Top 5 TypeScript Frameworks (2022).” *Mastering Backend*, 1 Mar. 2021, masteringbackend.com/posts/top-5-typescript-frameworks. Accessed 24 Feb. 2024.

Fetisov, Evgeniy, and Anastasiya Talochka. “Javascript vs. Typescript: Which Is Better for Your Project in 2023?” *JayDevs*, 6 Mar. 2023, jaydevs.com/javascript-vs-typescript/.

Nihar-Raval. “Typescript vs JavaScript: Which One Is Better to Choose?” *Radixweb*, Radixweb, 14 Dec. 2023, radixweb.com/blog/typescript-vs-javascript#difference.

“Frequently-Asked Questions | LoopBack Documentation.” *Loopback.io*, loopback.io/doc/en/lb4/FAQ.html. Accessed 24 Feb. 2024.

“JavaScript vs. TypeScript: What’s the Difference? | TheServerSide.” *TheServerSide.com*, www.theserverside.com/tip/JavaScript-vs-TypeScript-Whats-the-difference. Accessed 24 Feb. 2024.

Lee, Andrew. “Why (Most) Startups Should Only Write TypeScript.” *DEV Community*, 13 Nov. 2022, dev.to/andyrewlee/why-most-startups-should-only-write-typescript-2jam. Accessed 24 Feb. 2024.

Neel, Harish. “Why (Almost) Every Startup Should Only Use TypeScript.” *Medium*, 15 Feb. 2023,

medium.com/@harishneel/why-almost-every-startup-should-only-use-typescript-bafda593ffb5. Accessed 24 Feb. 2024.

Top 10 Startups Using TypeScript. 16 Feb. 2022, blog.back4app.com/top-10-startups-using-typescript/#Slack. Accessed 24 Feb. 2024.

"Ts.ED - a Node.js and TypeScript Framework on Top of Express/Koa.js." *Ts.ED - a Node.js and TypeScript Framework on Top of Express/Koa.js.*, tsed.io/getting-started/#philosophy. Accessed 24 Feb. 2024.

Martinez, Desiree, et al. "(PDF) a Review on Java Programming Language." *Research Gate*, 2023, www.researchgate.net/publication/371166744_A_Review_on_Java_Programming_Language.

Developer, Website. "Java Programming Language." *Medium*, Medium, 17 Mar. 2021, seattlewebsitedevelopers.medium.com/java-programming-language-c355f06ac12a#:~:text=Java%2C%20an%20object%2Doriented%20programming,here%20represent%20real%20words%20entity.

Allah, Zobair. "Research Article Role of JAVASCRIPT Function ..." *International Journal of Recent Scientific Research*, 2020, recentscientific.com/sites/default/files/15749-A-2020.pdf.

Bierman, Gavin, et al. *Understanding Typescript*, users.soe.ucsc.edu/~abadi/Papers/FTS-submitted.pdf. Accessed 25 Feb. 2024.

Hansson, David. "Turbo 8 Is Dropping TypeScript." David Heinemeier Hansson , Hey, 6 Sept. 2023, world.hey.com/dhh/turbo-8-is-dropping-typescript-70165c01. Accessed 24 Jan. 2024.

Harris, Rich. "Lordy, I Did Not Expect an Internal Refactoring PR to End up #1 on Hacker News. ... | Hacker News." Hacker News, May 2023, news.ycombinator.com/item?id=35892250. Accessed 24 Jan. 2024.

"Use JSDoc: Getting Started with JSDoc 3." Jsdoc.app, jsdoc.app/about-getting-started. Accessed 25 Jan. 2024.