To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub

Justus Bogner
justus.bogner@iste.uni-stuttgart.de
University of Stuttgart, Institute of Software Engineering
Stuttgart, Germany

Manuel Merkel
st155131@stud.uni-stuttgart.de
University of Stuttgart, Institute of Software Engineering
Stuttgart, Germany

ABSTRACT

JavaScript (JS) is one of the most popular programming languages, and widely used for web apps, mobile apps, desktop clients, and even backend development. Due to its dynamic and flexible nature, however, JS applications often have a reputation for poor software quality. While the type-safe superset TypeScript (TS) offers features to address these prejudices, there is currently insufficient empirical evidence to broadly support the claim that TS applications exhibit better software quality than JS applications.

We therefore conducted a repository mining study based on 604 GitHub projects (299 for JS, 305 for TS) with over 16M LoC. Using SonarQube and the GitHub API, we collected and analyzed four facets of software quality: a) code quality (# of code smells per LoC), b) code understandability (cognitive complexity per LoC), c) bug proneness (bug fix commit ratio), and d) bug resolution time (mean time a bug issue is open). For TS, we also collected how frequently the type-safety ignoring `any` type was used per project via ESLint.

The analysis indicates that TS applications exhibit significantly better code quality and understandability than JS applications. Contrary to expectations, however, bug proneness and bug resolution time of our TS sample were *not* significantly lower than for JS: the mean bug fix commit ratio of TS projects was more than 60% larger (0.126 vs. 0.206), and TS projects needed on average more than an additional day to fix bugs (31.86 vs. 33.04 days). Furthermore, reducing the usage of the `any` type in TS apps appears to be beneficial: its frequency was significantly correlated with all metrics except bug proneness, even though the correlations were of small strengths (Spearman's rho between 0.17 and 0.26).

Our results indicate that the perceived positive influence of TypeScript for avoiding bugs in comparison to JavaScript may be more complicated than assumed. While using TS seems to have benefits, it does not automatically lead to less and easier to fix bugs. However, more research is needed in this area, especially concerning the potential influence of project complexity and developer experience.

MSR 2022, May 23–24, 2022, Pittsburgh, PA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS CONCEPTS

• **Software and its engineering → Data types and structures**; **Language features**; **Software libraries and repositories**.

KEYWORDS

JavaScript, TypeScript, software quality, repository mining, GitHub

ACM Reference Format:
Justus Bogner and Manuel Merkel. 2022. To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub. In *Proceedings of MSR '22: Proceedings of the 19th International Conference on Mining Software Repositories (MSR 2022).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In 1995, JavaScript (JS) was introduced to add dynamic client-side functionality in the web browser [12]. With the growth of the Internet, its usage spread rapidly. According to a recent Stackoverflow survey[1], JavaScript is now the most widely used programming language. Potential reasons may include its versatility and ease of use. TypeScript, a superset of JavaScript, is also growing in popularity[2], with some claiming it will be *the* programming language for developing next-generation web apps, mobile apps, Node.js apps, and IoT software [9].

TypeScript (TS) mainly extends JavaScript with type annotations and provides `tsc`, the TypeScript Compiler to transpile TS to JS. TS performs type checking at transpile-time, while JS performs type checking at run-time [19]. This is one of the reasons why JavaScript does not have the best reputation for delivering high-quality software. Its beginner-friendliness and dynamic nature without a compiler lead some people to assume that using JavaScript often leads to poor software quality [11, 31], similar to sentiments about other scripting languages like PHP [1]. However, there is currently insufficient empirical evidence to support the claim that TypeScript leads to overall better software quality than JavaScript. The debate whether statically typed languages are better for software quality than dynamically typed ones has been going on for quite some time. Proponents of static typing insist that "it allows early detection of some programming errors" [27] and that it leads to "better documentation in the form of type signatures" [25]. Advocates of dynamic typing, however, claim that "it was easier to get things right with short source code, in which code that was not too terse or verbose determined behavior, when all types could

[1] https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-programming-scripting-and-markup-languages
[2] https://octoverse.github.com/#top-languages-over-the-years

Association for Computing Machinery, New York, NY, USA, 22–35. https://doi.org/10.1145/1869459.1869462
[15] Zofia Hanusz, Joanna Tarasinska, and Wojciech Zieliński. 2016. Shapiro–Wilk test with known mean. 14 (02 2016), 89–100.
[16] Ahmed E. Hassan. 2008. The Road Ahead for Mining Software Repositories. *Proceedings of the 2008 Frontiers of Software Maintenance, FoSM 2008*, 48–57. https://doi.org/10.1109/FOSM.2008.4659248
[17] Larry L. Havlicek and Nancy L. Peterson. 1976. Robustness of the Pearson Correlation against Violations of Assumptions. *Perceptual and Motor Skills* 43 (1976), 1319 – 1334.
[18] Glenn D. Israel. 1992. Determining Sample Size - GJIMT. https://www.gjimt.ac.in/wp-content/uploads/2017/10/2_Glenn-D.-Israel_Determining-Sample-Size.pdf
[19] Remo H. Jansen. 2015. *Learning TypeScript: exploit the features of TypeScript to develop and maintain captivating web applications with ease.* Packt Publishing. pp. 2 f. pages.
[20] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel German, and Daniela Damian. 2015. The Promises and Perils of Mining GitHub (Extended Version). *Empirical Software Engineering* (01 2015).
[21] Sebastian Kleinschmager, Romain Robbes, Andreas Stefik, Stefan Hanenberg, and Eric Tanter. 2012. Do static type systems improve the maintainability of software systems? An empirical study. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. 153–162. https://doi.org/10.1109/ICPC.2012.6240483
[22] Charles J. Kowalski. 1972. On the Effects of Non-Normality on the Distribution of the Sample Product-Moment Correlation Coefficient. 21, 1 (1972), 1. https://doi.org/10.2307/2346598
[23] W. Lenhard and A. Lenhard. 2016. Computation of effect sizes. https://doi.org/10.13140/RG.2.2.17823.92329
[24] Ronald P. Loui. 2008. In Praise of Scripting: Real Programming Pragmatism. *Computer* 41, 7 (jul 2008), 22–26. https://doi.org/10.1109/MC.2008.228
[25] Erik Meijer and Peter Drayton. 2004. Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages.
[26] Marvin Muñoz Barón, Marvin Wyrich, and Stefan Wagner. 2020. An Empirical Validation of Cognitive Complexity as a Measure of Source Code Understandability. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (Bari, Italy) *(ESEM '20)*. Association for Computing Machinery, New York, NY, USA, Article 5, 12 pages. https://doi.org/10.1145/3382494.3410636
[27] Benjamin C. Pierce. 2002. *Types and programming languages.* The MIT Press.
[28] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A Large Scale Study of Programming Languages and Code Quality in Github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 155–165. https://doi.org/10.1145/2635868.2635922
[29] Vlad Riscutia. 2019. *Programming with Types.* Manning, Birmingham.
[30] Gregorio Robles. 2010. Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories proceedings. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 171–180. https://doi.org/10.1109/MSR.2010.5463348
[31] Tobias Roehm, Daniel Veihelmann, Stefan Wagner, and Elmar Juergens. 2019. Evaluating Maintainability Prejudices with a Large-Scale Study of Open-Source Projects. In *Software Quality: The Complexity and Challenges of Software Engineering and Software Quality in the Cloud. SWQD 2019. Lecture Notes in Business Information Processing, vol 338.* Springer, Cham, 151–171. https://doi.org/10.1007/978-3-030-05767-1_10 arXiv:1806.04556
[32] Amir Saboury, Pooya Musavi, Foutse Khomh, and Giulio Antoniol. 2017. An empirical study of code smells in JavaScript projects. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 294–305. https://doi.org/10.1109/SANER.2017.7884630
[33] Shlomo S. Sawilowsky. 2009. New Effect Size Rules of Thumb. *Journal of Modern Applied Statistical Methods* 8, 2 (nov 2009), 597–599. https://doi.org/10.22237/jmasm/1257035100
[34] Emmit Scott. 2015. *SPA Design and Architecture: Understanding Single-Page Web Applications.* Manning Publications, Shelter Island, NY. 312 pages.
[35] Stefan Tilkov and Steve Vinoski. 2010. Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing* 14, 6 (nov 2010), 80–83. https://doi.org/10.1109/MIC.2010.145
[36] E. van Emden and Leon Moonen. 2012. Assuring software quality by code smell detection. xix–xix. https://doi.org/10.1109/WCRE.2012.69
[37] Aiko Yamashita. 2013. How Good Are Code Smells for Evaluating Software Maintainability? Results from a Comparative Case Study. In *2013 IEEE International Conference on Software Maintenance*. 566–571. https://doi.org/10.1109/ICSM.2013.97
[38] Jie Zhang, Feng Li, Dan Hao, Meng Wang, Hao Tang, Lu Zhang, and Mark Harman. 2020. A Study of Bug Resolution Characteristics in Popular Programming Languages. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/tse.2019.2961897
[39] Qimu Zheng, Audris Mockus, and Minghui Zhou. 2015. A Method to Identify and Correct Problematic Software Activity Data: Exploiting Capacity Constraints and Data Redundancies. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 637–648. https://doi.org/10.1145/2786805.2786866