The next step, 6G, is about more than just higher speeds; it's about creating intelligent and resilient systems that learn, adapt, and withstand enormous loads. Projects like SUSTAIN-6G, AMBIENT-6G, and 6G-VERSUS demonstrate a growing commitment to greener, fairer, and more sustainable technologies. If these efforts are successful, the networks of the future will not just transmit data but help shape a more connected world.

References

1. Budd, J. (2024). 5G redcap: Simplifying cellular connectivity for iot devices. ABI Research. https://www.abiresearch.com/blog/5g-redcap
2. Jones, D. (2024). Tesla deploys private 5G in its factory in Berlin. Fierce Network. https://www.fierce-network.com/wireless/tesla-goes-private-5g-its-factory-berlin
3. Tika, S., Haqiq, A., Sabir, E., & Driouch, E. (2025). Where 6G stands today: Evolution, enablers, and research gaps. arXiv.org e-Print archive. https://arxiv.org/html/2509.19646v1
4. University of Oulu. (2024). Five new 6G projects to drive sustainability, resilience, and inclusive innovation. 6G Flagship. https://www.6gflagship.com/news/6g-projects-sustainability-innovation

DOI 10.70286/EOSS-01.12.2025.009

EMPIRICAL STUDY OF THE IMPACT OF TYPESCRIPT ON CODE QUALITY AND DEVELOPMENT PRODUCTIVITY OF WEB APPLICATIONS

Kuznetsova Yuliia Anatoliivna
Ph.D., Associate Professor
Cherkasov Dmytro Yuriiovych
Master
National Aerospace University "Kharkiv Aviation Institute", Ukraine

Abstract. The impact of TypeScript, in comparison with JavaScript, on code quality and developer productivity is examined through the analysis of 85 projects (42 TypeScript, 43 JavaScript) and a controlled experiment with 60 developers. The results show that TypeScript leads to lower code-smell density and lower cognitive complexity, but requires more time to implement new features and produces slightly larger bundles. In refactoring tasks, it helps avoid a significant share of breaking changes and makes debugging noticeably faster.

Keywords: TypeScript; JavaScript; code quality; developer productivity; cognitive complexity; repository mining; refactoring.

Introduction. TypeScript is a statically typed superset of JavaScript developed by Microsoft in 2012. According to GitHub Octoverse 2024, TypeScript ranks third in

popularity among programming languages, and 78% of new React projects use TypeScript [1]. TypeScript is promoted as a way to improve code quality through static type checking and enhanced IDE support. A study by Bogner and Merkel (2022) on 604 GitHub projects showed that TypeScript applications exhibit better code quality and understandability, but, unexpectedly, have a higher bug-fix commit ratio (0.206 versus 0.126 for JavaScript) and longer bug-fix time (33.04 versus 31.86 days) [2]. This raises questions about the actual impact of static typing on productivity.

Aim and research objectives. The aim of the study is to comprehensively assess the impact of TypeScript on code quality and development productivity in web applications by combining repository mining with a controlled experiment involving developers. The research objectives are to:

1) perform repository-mining analysis of 85 open-source projects to measure static code-quality metrics;
2) conduct a controlled experiment with 60 developers to measure productivity;
3) compare code-smell density, cognitive complexity, and bug density between TypeScript and JavaScript;
4) measure the time for feature development, refactoring, and debugging in both languages;
5) analyze the impact of using the any type on TypeScript code quality;
6) evaluate bundle size and runtime performance overhead;
7) develop recommendations for choosing between TypeScript and JavaScript.

Research results and discussion. The study consisted of two parts: repository-mining analysis and a controlled experiment.

Repository Mining. A total of 85 active open-source projects from GitHub were selected (42 TypeScript, 43 JavaScript) according to the following criteria: at least 1000 stars, active development during the last six months, and at least 10 contributors. The total volume of analyzed code amounted to 2,847,392 lines (TypeScript: 1,423,156 LoC; JavaScript: 1,424,236 LoC). The analysis used SonarQube 9.9, ESLint 8.5, and TypeScript Compiler 5.3. Cognitive complexity was measured according to the methodology described in [3]. Three types of tasks were performed on an identical e-commerce code base: implementation of a wishlist feature, refactoring of a large component, and fixing five bugs. The results of repository mining are presented in Table 1.

Table 1 – Comparative code-quality metrics for TypeScript and JavaScript

Metric	TypeScript	JavaScript	Difference
Code smells per 1000 LoC	12.7	19.3	−34%
Cognitive complexity per LoC	0.0084	0.0117	−28%
Bug fix commit ratio	0.189	0.143	+32%
Average bug-fix time (days)	29.4	26.8	+10%
Security vulnerabilities per 1000 LoC	0.8	1.3	−38%
Use of any in TypeScript	18.7%	0%	−18.7%
Production bundle size (KB)	287	266	+8%

*data obtained through analysis of 85 GitHub projects and compiled by the authors

TypeScript demonstrated markedly better static code-quality indicators: 34% fewer code smells and 28% lower cognitive complexity. This is consistent with the findings of Bogner and Merkel [2]. At the same time, the bug-fix commit ratio was 32% higher, and bug-fix time 10% longer. A plausible explanation is that TypeScript catches many defects at compile time, so the bugs that reach runtime tend to be more complex [4].

Analysis of any usage showed that, on average, 18.7% of TypeScript code uses any. Correlation analysis revealed that projects with any usage above 25% have code-quality metrics close to those of JavaScript projects, which confirms the importance of disciplined use of typing [5].

The controlled experiment involved 60 developers with at least two years of JavaScript experience. Participants were randomly assigned to two groups (TypeScript and JavaScript) and solved three types of tasks on an identical code base: development of a new feature, refactoring of an existing component, and fixing five bugs. The results are presented in Table 2.

Table 2 – Comparative metrics of developer productivity

Metric	TypeScript	JavaScript	Difference
Development of a new feature:			
Development time (min)	89.3	79.6	+12%
Compilation errors	14.2	3.1	+358%
Runtime errors	2.3	7.8	−71%
Number of refactors	5.7	8.4	−32%
Code refactoring:			
Refactoring time (min)	34.7	28.2	+23%
Result quality (0-10)	8.4	6.9	+22%
Number of breaking changes	1.8	5.2	−65%
Bug fixing:			
Time to first fix (min)	11.8	11,8	−23%
Total time (min)	38.4	42.7	−10%
Correctly fixed bugs	4.7	4.2	+12%
NASA-TLX (points)	52.3	48.6	+8%

*data obtained from an experiment with 60 developers and compiled by the authors

Developers using TypeScript spent 12% more time on a new feature due to the need to define types and fix 14.2 compilation errors (versus 3.1). However, runtime errors decreased by 71%. In refactoring tasks, the time increased by 23%, but the number of breaking changes decreased by 65% thanks to static checking [6]. The quality of the result was 22% higher. In bug-fixing tasks, TypeScript enabled a 23% faster first fix due to IDE support (IntelliSense). The total time decreased by 10%, and more bugs were fixed correctly (4.7 vs 4.2 out of five) [7].

Cognitive load (NASA-TLX) was 8% higher for TypeScript. Participants commented that «types slow you down at first but save time on debugging».

Correlation analysis showed that the benefits of TypeScript are most pronounced in large projects (>50,000 LoC) and teams of five or more developers. For small projects (<10,000 LoC), the overhead from typing may outweigh the advantages [8].

Conclusions. The empirical study confirmed that TypeScript provides substantial improvements in static code quality: 34% fewer code smells, 28% lower cognitive complexity, and 38% fewer security vulnerabilities. At the same time, its impact on development productivity is mixed.

TypeScript increases initial development time by 12% because of type definitions, but reduces runtime errors by 71% and speeds up debugging by 23%. In refactoring tasks, TypeScript results in 65% fewer breaking changes, which is important for the maintenance of large code bases.

A key factor in the effectiveness of TypeScript is disciplined use of typing. Projects with more than 25% usage of any lose much of the benefit of static typing. TypeScript is most effective for large projects (>50,000 LoC), long-term maintenance, and teams of five or more developers.

Practical recommendations are to use TypeScript for enterprise applications and long-lived projects; avoid excessive use of the any type (keep it below 15%); invest in team training on TypeScript best practices; enable strict mode in tsconfig.json; use ESLint with @typescript-eslint/recommended; and, for small projects (<10,000 LoC), consider JSDoc as an alternative.

References

1. GitHub. (2024). The State of the Octoverse 2024: Top programming languages. GitHub Blog. https://github.blog/news-insights/octoverse/octoverse-2024/.
2. Bogner, J., & Merkel, M. (2022). To type or not to type? A systematic comparison of the software quality of JavaScript and TypeScript applications on GitHub. In Proceedings of the 19th International Conference on Mining Software Repositories (MSR ‘22) (pp. 658–669). ACM. https://doi.org/10.1145/3524842.3528454.
3. Muñoz Barón, M., Wyrich, M., & Wagner, S. (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 ’20) (Article 5). ACM. https://doi.org/10.1145/3382494.3410686.
4. Wang, N., Chen, T., & Xu, B. (2025). An empirical study on bugs in TypeScript programming language. Journal of Systems and Software, 222, 112445. https://doi.org/10.1016/j.jss.2025.112445.
5. Emmanni, P. S. (2021). The role of TypeScript in enhancing development with modern JavaScript frameworks. International Journal of Science and Research, 10(2), 1738–1741. https://doi.org/10.21275/SR24401234212.
6. Kim, M., Zimmermann, T., & Nagappan, N. (2014). An empirical study of refactoring challenges and benefits at Microsoft. IEEE Transactions on Software Engineering, 40(7), 633–649. https://doi.org/10.1109/TSE.2014.2318734.

7. Silva, D., Tsantalis, N., & Valente, M. T. (2024). Refactoring React-based web apps. Journal of Systems and Software, 214, 112065. https://doi.org/10.1016/j.jss.2024.112065.
8. Mendonça, W. L., Santos, J. A., Ferreira, K. A., & Valente, M. T. (2025). Understanding the adoption of modern JavaScript features: An empirical study on open-source systems. Empirical Software Engineering, 30(2), Article 42. https://doi.org/10.1007/s10664-025-10663-9.

THE USE OF ARTIFICIAL INTELLIGENCE FOR CYBERATTACK PREDICTION IN CORPORATE NETWORKS

Serheiev Artem
cadet
Makalish Bohdan
cadet
Podvalnyi Arsenii
cadet
Department of Cybercrime Counteraction
Kharkiv National University of Internal Affairs, Ukraine

In today's digital world, information security has become one of the key factors determining the stability and development of enterprises. The constant growth in the number, complexity, and scale of cyberattacks requires specialists to apply innovative methods of protecting corporate networks. Traditional security systems based on signature analysis or manual policy configuration no longer provide an adequate level of protection. Therefore, increasing attention is being paid to approaches that utilize artificial intelligence (AI) and machine learning methods for the detection**,** analysis**,** and prediction of cyberattacks [1, p. 2].

The main advantage of using artificial intelligence lies in its ability to learn from vast amounts of data and identify hidden patterns in network traffic behavior. Machine learning algorithms enable the creation of models that analyze user and device activity in real time and can predict potential threats before they occur [2, p. 3]. For instance, systems based on deep neural networks can recognize abnormal connections or uncharacteristic user actions that may indicate an ongoing attack.

It should be noted that the effectiveness of such systems largely depends on the quality of the data used for training. According to the FAIR principles (Findability, Accessibility, Interoperability, Reusability), which are being actively implemented in the field of research data management [2, p. 2], it is critically important to ensure data accessibility, compatibility, and reusability. This enhances the accuracy of models and