On the Usage of New JavaScript Features through Transpilers: The Babel Case

Thiago Nicolini, Andre Hora, Eduardo Figueiredo

Department of Computer Science, UFMG, Brazil
{tnicolini, andrehora, figueiredo}@dcc.ufmg.br

Abstract

JavaScript applications have long been dealing with browser compatibility issues. Due to the variety of browsers, operational systems, and versions, not all new JavaScript features are available to users. Fortunately transpilers transform the new code syntax into an older version, making the code compatible with a large number of browsers. This study, which focuses on a largely used transpiler (Babel), aims to shed light on how new JavaScript features are used and what major challenges developers face when using them. We assess the top-1K JavaScript projects from GitHub, 108 Stack Overflow questions, and the browser compatibility related to 18 recent JavaScript features. We conclude that, without a transpiler, almost 14% of the web users could face a JavaScript bug when landing on a website that uses a new JavaScript feature.

Keywords— JavaScript, Transpilers, Babel, Browser compatibility, Software maintenance

1 Introduction

Browser compatibility is still a challenge in modern software development. Even with two major players (Chrome and Safari) representing more than 80% of the browser market share worldwide, seven other browsers are responsible for 16% of the Web traffic.[1] This profusion of options for Web browsing shows that the user has a lot of flexibility [7], but it also indicates that developers have to be careful to avoid cross-browser compatibility (CBC) issues. Indeed, the CBC problem is almost as old as the Web browser itself [8]. These issues are a major concern when developers want to use the latest features of JavaScript.[2]

[1] `https://gs.statcounter.com/browser-market-share`
[2] `https://tinyurl.com/2p9f6673`

JavaScript is a programming language that follows the ECMAScript specification, created by the ECMA International [5], an association that defines the standards for technologies. The ECMA Technical Committee 39 (TC39) is in charge of evaluating proposals for additional features in the ECMAScript standard. The idea of a proposal is adding specification of new features that improve the JavaScript development.[3] To became a standard feature, the proposal has to pass trough five stages process.[4] When a feature is under the last two stages, browsers can implement their specification to make it available to developers.

Every year the TC39 releases a new ECMAScript version but not all browsers provide full compatibility to it. In order to use new JavaScript features, developers often rely on JavaScript transpilers.[5] A transpiler is a tool that takes the source code and generates code written in another target language, which is syntactically equivalent to the original source. Web developers rely on JavaScript transpilers to use proposals even in the early stages. These tools convert the proposed novel feature into an old JavaScript syntax. In this scenario, we can highlight the Babel transpiler (https://babeljs.io). Babel is a transpiler used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript. This NPM package is the most popular transpiler in the JavaScript ecosystem with an average of 32 million weekly downloads.[6]

Despite the fact that the use of new JavaScript features is present in major open-source projects (such as, VueJS, React, and JQuery), some questions remain unanswered. For example, how frequently new JavaScript features are adopted in open-source projects? What challenges developers face when using new JavaScript features? How compatible are the new JavaScript features in current browsers? Answering those questions would provide the basis to better understand the usage landscape of new JavaScript features and reveal the possible impact on current browsers.

In this paper, we propose a quantitative and qualitative empirical study to explore the usage of new JavaScript features through the Babel transpiler. *First*, we analyze 1,000 popular JavaScript open-software projects, searching for the use of Babel plugins that indicates the adoption of new JavaScript features. *Second*, we assess the Stack Overflow question dataset to understand the issues developers face when using these plugins. *Third*, we explore the *Can I Use* platform [3], which provides up-to-date browser compatibility for front-end Web technologies on desktop and mobile browsers. We analyze the compatibility of the new JavaScript features across the most used Web browsers (Chrome, Safari, Firefox, and Edge). We then propose the following research questions:

- **RQ1**: How often new JavaScript features are used in software projects?
- **RQ2**: What issues are faced by developers when using plugins to transpile the new JavaScript features?

[3] `https://tinyurl.com/4bf6yhab`
[4] `https://tc39.es/process-document`
[5] `https://tinyurl.com/3xxn5cpf`
[6] `https://www.npmjs.com/package/@babel/core`

- **RQ3**: What is the impact of new JavaScript features on browser compatibility?

Major findings. We find that developers commonly rely on the Babel transpiler to handle new JavaScript features. We detect that 73 projects (out of 1,000) use at least one new JavaScript proposal (RQ1). We also detect that developers struggle to import and use the transpiler proposal plugins into their projects (RQ2). Finally, we discover that new JavaScript features have an average of 86% of compatibility across the most common browsers (RQ3). These findings suggest that the transpilers play a relevant role in the modern Web development. Without them, the usage of the new JavaScript features would be in jeopardy because 14% of the global users could face CBC issues when accessing a website with these new features.

Contributions. The contributions of this paper are twofold. First, we provide a large empirical study to better understand the usage and challenges of new JavaScript features. Second, we discuss practical implications regarding the impact of new JavaScript features on browser compatibility. Our dataset is publicly available at: `https://zenodo.org/record/7250577`.

2 Study Design

This section describes the study design to address RQ1, RQ2, and RQ3.

2.1 Mining projects that rely on Babel (RQ1)

To understand how frequent is the use and the browser compatibility of new JavaScript features, we analyze the top-1K JavaScript open-source projects hosted on GitHub to search for projects that use the Babel transpiler. We first select 1,000 JavaScript projects found on GitHub, sorted by the number of stars, which is a proxy of popularity [1]. We rely on the GitHub search API,[7] querying for JavaScript projects and sorting by the number of stars. This search provides 1,000 results, which is the number of repositories analysed. We then search for projects using Node Package Manager (NPM) to import Babel modules. For this purpose, we develop a script to go over all project's repositories searching for *package.json*, which records the project's dependencies. Then, we parse these *package.json* files, searching for all occurrences of the *babel* token. To avoid false positives, we assess whether the package found is indeed a package from Babel documentation.[8] Lastly, to ensure that the project is really using the Babel package (and it is not only importing it via the *package.json*), we rely on the Depcheck[9] tool to check if a dependency is used in code. Specifically, we randomly select 182 projects (95% confidence level and 5% confidence interval) and run the Depcheck on those projects. We found that 98% of the projects that imports Babel modules are indeed using them.

[7] `https://tinyurl.com/34mkekhu`
[8] `https://babeljs.io/docs/en/`
[9] `https://github.com/depcheck/depcheck`

2.2 Exploring StackOverflow questions (RQ2)

Next, we rely on the Stack Overflow Search API[10] to find the challenges developers face to adopt a new JavaScript feature. Since these features are at the proposal level of the TC39 Committee Process, the Babel plugins to transpile these new features into old JavaScript syntax have the *proposal* prefix in their names. Thus, we use Stack Overflow API to search for all questions that include the term *babel/plugin-proposal*. This pattern identifies all proposal plugins available in Babel documentation.[11] This way, we find 125 questions that match the criteria. To filter out false positives, we manually inspected each returned question. First, we check if a *babel/plugin-proposal* term found in a question is indeed a package from Babel documentation. After, we check if the question have the term *plugin-proposal*, but are related to another subject. For example, the developer may have simply pasted the *package.json* file (with few occurrences of *babel-plugin-proposals*), but ask about another dependency. After applying the filters, we remain with 108 valid questions. To analyze the developer's questions qualitatively, we rely on *thematic analysis* [4], which consists in a procedure to identify, analyze, and report themes that are present in the data of qualitative research. Thus, we aim to identify and record themes in textual documents, using the following steps: (1) initial reading of the questions, (2) generating a first code for each question, (3) searching for themes among the proposed codes, (4) reviewing the themes to find opportunities for merging, and (5) defining and naming the final themes. The first three steps were performed by the first author of the paper, while steps 4 and 5 were done together by all authors of the paper until consensus was achieved. Thematic analysis is also used in the following two steps.

2.2.1 Grouping questions by development cycle phase

To understand in which phase of the development cycle developers face issues with transpilers plugins, we use thematic analysis to group the questions in three categories. **Project Setup:** when the question is related to import the plugin, setting up a development environment (e. g., *Trouble Installing babel's plugin-proposal-export-default-from*). **Development:** when the question is related to the plugin usage itself, with code examples, asking about syntax, or how to integrate to existing code (e. g., *How do I decorate an async method using Babel 7's plugin-proposal-decorators with an async function?*). **Build:** when the question is related to publishing the application to the Web, build and bundle errors (e. g., *Compilation error when building ionic app*). We found three questions about testing. The first two were related to plugin importing, thus we grouped them into the **Project Setup** phase. The third one was related to the usage of new JavaScript features in unit testing code. Since it was related to the writing code process, we grouped it into the **Development Phase**. Based on this classification, we aim to identify in which phase of development the transpiler usage is more challenging.

[10] `https://api.stackexchange.com`

[11] `https://babeljs.io/docs/en/plugins-list`

2.2.2 Classifying the questions in topics

To understand the major problem the developer is facing when using the proposal plugin, we classify the questions in four topics. **Test issues:** questions related to a test suite not working properly due to Babel (e.g., *How can I enable decorators support when running tests with CRA 2.1?*). **IDE miss configuration:** questions related to an error in an IDE/editor due to the plugin usage (e.g., *Visual Studio Code error messages and Babel plugins*). **Incorrect plugin importing:** questions related to how properly import plugins, dependency managers (like npm and webpack), and importing other dependencies; (e.g., *Add plugin-proposal-class-properties to create-react-app project*). **Plugin miss usage:** questions related to the plugin usage in general, like code syntax, and integration with existing code (e.g., *Babel plugin-proposal-decorators not working as expected*).

2.3 Assessing browser compatibility (RQ3)

In this last analysis, we rely on the *Can I Use* platform [3] to assess the compatibility of the proposal plugin with the most popular browsers. This platform provides up-to-date browser compatibility for front-end Web technologies on desktop and mobile browsers [3]. It has data about the version of a browser that supports a given feature and a global estimation (in percentage) about that feature. For example, *Arrow Functions* [3], which became a JavaScript feature in the ES6 realease (2015), has an estimate of 96.11% of global compatibility. This platform also indicates that the feature became available in Chrome Desktop in the version 45. Although the platform already provides an upfront useful data, unfortunately, it does not make it clear the total browser versions that support JavaScript features.

To overcome this limitation, we develop a script to compute both (i) the total number of browser versions and (ii) the versions that support JavaScript features based on the *Can I Use* data. We are then able to assess the number of browser versions available and the number of versions that supports the proposal features. For example, a browser X may have a total of 100 versions, but only 10 versions support a given JavaScript feature. Our final goal is to better understand to what extent the transpiler plugins play a role in making the JavaScript releases available for the overall public.

We select all 30 available Babel plugins that are related to proposal functionalities (new JavaScript features), which means the available plugins with the notation *babel/plugin-proposal-<plugin name>*. Then, we check their compatibility rate in the *Can I use* platform [3]. This compatibility rate is an estimation between the global internet traffic and the browser/version usage.

3 Results

This section describes the results of this research.

3.1 RQ1: Usage of New JS features

By mining the top-1K JavaScript projects, we find 1,135 occurrences of Babel plugins in 345 different projects. Thus, we identify that close to 35% of the top-1k JavaScript projects on GitHub projects rely on Babel transpilers in some way. The most common plugin occurrence is *core* (282 occurrences). This is expected because this module has the main transpiler methods, like the *transform* which is the method responsible for transform the source code.[12]

We now focus on analyzing the most used proposal plugins, that is, the plugins that indicate the usage of new JavaScript features. As summarized in Table 1, we found 144 proposal occurrences, with 16 distinct plugins being used by 73 projects. The most used plugin is Class Properties (54), which transpiles class properties (such as constructor, static fields, and methods) into object properties since object properties have better compatibility among browsers. The second most used plugin is Object Rest Spread (29), which transpiles the spread operator ... into Object assigned properties. We also identify the presence of Babel proposal plugins in some well-known JavaScript open-source projects, such as VueJS and ReactJS. Overall, 7.3% (73 out of 1,000) of all studied projects use a proposal plugin. Since other software projects rely on these widely popular frameworks as a dependency, indirectly, proposals are vastly used in Web development.

Table 1: Frequency of Proposal Babel plugins.

Proposal Name	#Frequency
plugin-proposal-class-properties	54
plugin-proposal-object-rest-spread	29
plugin-proposal-optional-chaining	11
plugin-proposal-nullish-coalescing-operator	10
plugin-proposal-export-default-from	9
plugin-proposal-decorators	8
plugin-proposal-export-namespace-from	6
plugin-proposal-numeric-separator	3
plugin-proposal-optional-catch-binding	3
plugin-proposal-do-expressions	2
plugin-proposal-function-sent	2
plugin-proposal-pipeline-operator	2
plugin-proposal-throw-expressions	2
plugin-proposal-async-generator-functions	1
plugin-proposal-function-bind	1
plugin-proposal-logical-assignment-operators	1
Total	144

[12] `https://babeljs.io/docs/en/babel-core`

RQ1 summary: Babel has a relevant presence in the JavaScript ecosystem: 35% of the studied JavaScript projects rely on the Babel plugins. Among those projects, we found that 73 rely on a proposal plugin indicating the usage of a new JavaScript feature. The most commonly used new JavaScript feature is Class Properties.

3.2 RQ2: Issues when using new JS features

Considering the 108 analyzed questions from Stack Overflow, we found 12 different proposal plugins. The *plugin-proposal-class-properties* is the most common occurrence, representing 40% of the sample. This result is somehow expected since this plugin is the most common proposal plugin, as presented in RQ1. Figure 1 summarizes the distribution of questions by topic and phase. *Incorrect plugin importing* (56) and *Plugin miss usage* (40) are the most common issues developers face. Next, we have issues related to *IDE miss configuration* (8) and *Test* (4). Also, 58% of the analyzed issues are related to the *Development* phase: we find a total of 48 questions regarding new JavaScript features implementation, examples, and code syntax. We also find a large number of questions (37%, 40 questions) in the *Project Setup* phase. It indicates that developers are willing to use proposals in their projects, but, they may be failing to set up these features. Lastly, issues are less common in the *Build* phase. That can be explained because if developers are facing issues in the previous phases, they probably do not reach the build phase.

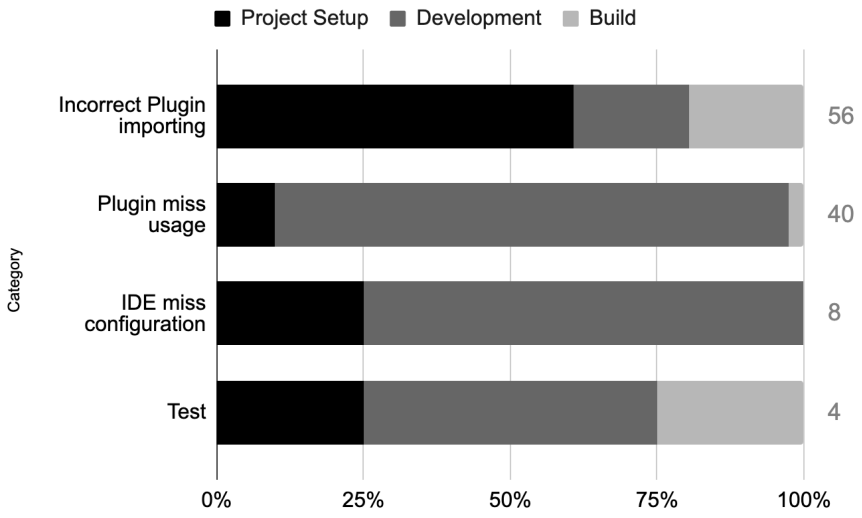

Figure 1: Stack Overflow Questions by topic and phase.

Table 2: New JavaScript Features *vs.* Browsers Compatibility

Proposal Name	Supported Desktop Versions (286 versions analysed)	Supported Mobile Versions (14 versions analysed)	Compatibility Rate (%)
unicode-property-regex	161 (56%)	14 (100%)	94.32
object-rest-spread	130 (45%)	10 (71%)	93.48
async-generator-functions	123 (43%)	9 (64%)	93.25
optional-catch-binding	115 (40%)	10 (71%)	93.25
syntax-import-meta	113 (39%)	10 (71%)	93.23
json-strings	96 (34%)	14 (100%)	92.76
numeric-separator	85 (30%)	12 (86%)	91.79
nullish-coalescing-operator	69 (24%)	13 (93%)	90.86
optional-chaining	69 (24%)	12 (86%)	90.86
logical-assignment-operators	69 (24%)	14 (100%)	89.17
class-static-block	66 (23%)	14 (100%)	88.98
class-properties	63 (22%)	14 (100%)	88.57
syntax-bigint	60 (21%)	14 (100%)	76.69
export-default-from	77 (27%)	5 (36%)	75.44
export-namespace-from	77 (27%)	5 (36%)	75.44
private-methods	40 (14%)	6 (43%)	74.00
syntax-top-level-await	33 (12%)	6 (43%)	73.24
private-property-in-object	19 (7%)	4 (29%)	70.08
Average	**81 (28%)**	**10 (70%)**	**85.86**

RQ2 summary: The major challenge to adopt a new JavaScript feature via the transpiler plugins are related to *incorrect plugin importing* (56) and *plugin miss usage* (40). Moreover, most of the questions are related to the *development* phase of the software life cycle.

3.3 RQ3: Browser compatibility

We found data for 18 out of the 30 available Babel proposal plugins. Since *Can I use* [3] is a open source platform, it relies on the number of developers requests (on GitHub) to present data of new JavaScript feature. We analyze information regarding 286 desktop browser versions, with following distribution: 73 versions of Opera, 14 of Safari, 80 of Chrome, 8 of Internet Explorer, 91 of Firefox and 20 of Edge. Furthermore, we analyse 14 versions of mobile browsers, with the following distribution: 2 of Chrome Mobile, 2 of Firefox Mobile, 2 of Samsung mobile, and 8 of Safari mobile.

Table 2 presents the results by desktop versions, mobile versions, and the total of compatibility rate of a given proposal plugin. Overall, we found that a new JavaScript feature has close to 86% of compatibility across the browsers. Moreover, mobile browsers have a better compatibility rate than desktop ones. The feature we identified as the most frequent on GitHub projects in RQ1 (*class-properties*) has close to 88% of compatibility. This highlights the relevance of the transpiler plugin when using this new feature. Making 12% of the global traffic unable to access the application due to compatibility issues is certainly a major concern. Some older features like *export-namespace-from* (ES2020) has a smaller compatibility rate (75.44%) when compared to newer features like *numeric-separator* from ES2021 (91.79%). This may be explained by the complexity of the feature and how browsers will interpret it.

RQ3 summary: Overall, a new JavaScript feature has 86% of compatibility across the browsers. Mobile browsers have a better compatibility rate when compared to desktop ones.

4 Discussion and Implications

This section discusses implications for both practitioners and researchers based on our results.

4.1 For Practitioners

Novel empirical data on the usage of new JavaScript features. By assessing the Babel adoption in the wild, we find new JavaScript features being used in 73 out of top-1K most popular JavaScript projects (RQ1). This includes the usage by well-know open-source projects, such as React and VueJS. Since these frameworks are commonly used in other projects, it indicates that these new features are indirectly present in a higher number of applications. Just to illustrate this scenario, according to NPM data, React has 11,701,844 weekly downloads.[13] We thus shed some light on the usage of new JavaScript features via the usage of the Babel transpiler.

Guidelines to support the usage of proposal plugins. The major challenge when adopting a new JavaScript feature (52%) is related to how properly import a proposal plugin (RQ2). This suggests that developers do want to use new features, leveraging compilers to address browser compatibility. However, due to the possible lack of documentation, lack of knowledge, or even incompatibility with other dependencies present in the JavaScript Projects, programmers may struggle to set up the feature in their projects, and, consequently, not use the new JavaScript feature. In this context, guidelines can be proposed to aid developers in properly importing the proposal plugins.

4.2 For Researchers

Factors that influence the adoption of a new JavaScript feature. In RQ3, we have seen that developers may adopt a newer JavaScript feature before adopting an older one. For example, some features available since 2018 are not commonly used like the *optional-chaining*, which is part of ES2020. In this scenario, researchers can investigate the factors that influence the adoption of a new feature. This can, for instance, shed light on which type of features should be prioritized in the upcoming JavaScript releases.

The practical benefits of using new JavaScript features. We detect that well-known software projects rely on new JavaScript features (RQ1). Novel studies can focus on identifying the benefits

[13] https://www.npmjs.com/package/react

of the adoption of new JavaScript features. For example, one could compare the application performance before and after the adoption. This type of study can contribute to attract early adopters for new JavaScript features, helping to advertise their usage among JavaScript community.

5 Threats to Validity

In this study, we assessed the presence of Babel plugins in the top-1K star ranked JavaScript GitHub projects. These projects are relevant projects, however, they may not represent the whole population of JavaScript. We also focused on projects that rely on NPM as a package manager tool because it is the most common tool for this purpose. Thus, if the project used another package manager, it was not part of the scope of the paper. Regarding the categorization of the Stack Overflow questions in RQ2, it is subjected to human bias in the classification. To overcome this possible threat, we relied on thematic analysis [4]. Finally, by focusing on Babel made it possible to analyze JavaScript transpilers from different data sources (GitHub, Stack Overflow, and Can I Use). We opt for addressing Babel because it is the most common transpiler for JavaScript projects. However, further research is necessary to address other transpilers.

6 Related Work

Mesbah *et al.* [8] defines how cross-browser issues impact Web software development. The authors proposed an automated method for cross-browser compatibility testing Web applications. Guoquan Wu *et. al.* [14] proposed a technique to detect cross-browser issues for JavaScript-based Web applications. Xu and Zeng [15] proposed a technique for statically analyzing cross-browser compatibility problems. In this paper, we also explore browser compatibility, taking a deeper look on the adoption of new JavaScript features.

Regarding transpilers, Kimura *et. al.* [6] proposed *Escapin*, a JavaScript transpiler for developing application programs that consume APIs and are deployed on cloud services to obtain new business concepts by trial-and-error iterations. Sayed *et al.* [12] proposed a JavaScript transpiler, which is an instrumented version with the flow-sensitive security monitor inlined. Our study focuses on the usage of Babel, which is a largely used JavaScript transpiler by the developer community.

Finally, JavaScript has been the focus of software engineering research for long period [2, 9, 10, 13]. Similarly, the Stack Overflow dataset is a important source of data for researches nowadays [11]. In summary, the related literature tackles different browser compatibility issues and proposes distinct tools to handle this issue. Other studies describe how transpilers work in general, with different applicability and in distinct programming languages. However, none of them is focused on the new JavaScript features and their impact on cross-browser issues.

7 Conclusion

This paper presented an empirical study on the usage of new JavaScript features and their impacts on browser compatibility. We took a deeper look at a JavaScript transpiler, and how it allows a developer to use new JavaScript releases even though these are not yet supported by the majority of browsers. As future work, we plan to conduct qualitative study to better understand *how* and *why* developers decide to use a new JavaScript release in their projects. We plan to conduct interviews and surveys to address these questions. Lastly, we aim to understand how new JavaScript features can be used in a faster and widely spread way.

References

[1] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of github repositories. In *International Conference on Software Maintenance and Evolution*, pages 334–344. IEEE, 2016.

[2] Aline Brito, Andre Hora, and Marco Tulio Valente. Characterizing refactoring graphs in Java and JavaScript projects. *Empirical Software Engineering*, 26(6):1–43, 2021.

[3] Can I Use? https://caniuse.com, 2021. [Online; accessed October 2022].

[4] D. S. Cruzes and T. Dyba. Recommended steps for thematic synthesis in software engineering. In *International Symposium on Empirical Software Engineering and Measurement*, pages 275–284, 2011.

[5] ECMA International. https://www.ecma-international.org, 2021. [Online; accessed October 2022].

[6] K. Kimura, A. Sekiguchi, S. Choudhary, and T. Uehara. A JavaScript Transpiler for Escaping from Complicated Usage of Cloud Services and APIs. In *Asia-Pacific Software Engineering Conference*, pages 69–78, 2018.

[7] X. Li and H. Zeng. Modeling web application for cross-browser compatibility testing. In *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing)*, pages 1–5, 2014.

[8] A. Mesbah and M. R. Prasad. Automated cross-browser compatibility testing. In *International Conference on Software Engineering*, pages 561–570, 2011.

[9] Romulo Nascimento, Eduardo Figueiredo, and Andre Hora. JavaScript API Deprecation Landscape: A Survey and Mining Study. *IEEE Software*, 39(3):96–105, 2022.

[10] Katerina Paltoglou, Vassilis E. Zafeiris, N.A. Diamantidis, and E.A. Giakoumakis. Automated refactoring of legacy JavaScript code to ES6 modules. *Journal of Systems and Software*, 181:111049, 2021.

[11] Riccardo Rubei, Claudio Di Sipio, Phuong T. Nguyen, Juri Di Rocco, and Davide Di Ruscio. PostFinder: Mining Stack Overflow posts to support software developers. *Information and Software Technology*, 127:106367, 2020.

[12] B. Sayed, I. Traoré, and A. Abdelhalim. If-transpiler: Inlining of hybrid flow-sensitive security monitor for javascript. *Computers & Security*, 75:92–117, 2018.

[13] F. Silva, H. Borges, and M. T. Valente. On the (un-)adoption of JavaScript front-end frameworks. *Software: Practice and Experience*, 1:1–27, 2021.

[14] G. Wu, M. He, H. Tang, and J. Wei. Detect cross-browser issues for javascript-based web applications based on record/replay. In *International Conference on Software Maintenance and Evolution*, pages 78–87, 2016.

[15] S. Xu and H. Zeng. Static Analysis Technique of Cross-Browser Compatibility Detecting. In *International Conference on Applied Computing and Information Technology*, pages 103–107, 2015.