UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Thiago Nicolini

Assessing the Usage of New JavaScript Features: A Survey and Mining Study

Belo Horizonte
2022

Thiago Nicolini

Assessing the Usage of New JavaScript Features: A Survey and Mining Study

Final Version

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: André Hora
Co-Advisor: Eduardo Figueiredo

Belo Horizonte
2022

Silva,Thiago Augusto Nicolini.

S586a Assessing the usage of new Javascript features[recurso eletrônico]: a survey and mining study / Thiago Augusto Nicolini Silva – 2022.
1 recurso online (55 f. il, color.): pdf.

Orientador: André Cavalcante Hora.
Coorientodor: Eduardo Magno Lages Figueiredo.
Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 52-55.

1. Computação – Teses. 2. JavaScript (Linguagem de programação de computador) – Teses. 3. Browsers (Programas de computador)- Compatibilidade – Teses. 4. Software – Manutenção – Teses. I. Hora, André Cavalcante. II. Figueiredo, Eduardo Magno Lages. III. Universidade Federal de Minas Gerais; Instituto de Ciências Exatas, Departamento de Ciência da Computação. IV. Título.

CDU 519.6*32(043)

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa
CRB 6/1510 Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

ASSESSING THE USAGE OFNEW JAVASCRIPT FEATURES: A SURVEY AND MINING STUDY

THIAGO AUGUSTO NICOLINI SILVA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. André Cavalcante Hora - Orientador
Departamento de Ciência da Computação - UFMG

Prof. Eduardo Magno Lages Figueiredo
Departamento de Ciência da Computação - UFMG

Prof. Marco Tulio de Oliveira Valente
Departamento de Ciência da Computação - UFMG

Prof. Rafael Serapilha Durelli
Departamento de Ciência da Computação - Universidade Federal de Lavras

Belo Horizonte, 15 de dezembro de 2022.

Superior, em 25/01/2023, às 14:33, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Marco Tulio de Oliveira Valente**, **Professor do Magistério Superior**, em 08/02/2023, às 14:44, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Eduardo Magno Lages Figueiredo**, **Professor do Magistério Superior**, em 08/02/2023, às 15:54, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Rafael Serapilha Durelli**, **Usuário Externo**, em 15/06/2023, às 11:08, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2040185** e o código CRC **98AFE5E3**.

Referência: Processo nº 23072.203940/2023-13 SEI nº 2040185

Dedico este trabalho a Vanessa e aos meus familiares.

Acknowledgments

Agradeço aos meus familiares, amigos e professores. Agradeço em especial:

A minha esposa Vanessa, professora do curso de Relações Públicas da UFMG, por ser a maior incentivadora para eu buscar a titulação em uma área na qual não sou egresso. Esse trabalho não seria possível sem o seu suporte.

Aos meus familiares, pelo incentivo e suporte na conclusão em mais esta etapa, principalmente meus pais, Marinês e Edimar, que sempre me preconizaram a dedicação aos estudos.

Aos meus sogros, pelo incentivo e suporte na realização do mestrado.

Ao meu orientador, Prof. André Hora, pela dedicação e pelos ensinamentos durante todo o mestrado.

Ao meu coorientador, Prof. Eduardo Figueiredo, pela participação conjunta nas orientações e por todas as constribuições neste processo.

Aos membros da banca, Prof. Marco Tulio Valente e Prof. Rafael Serapilha Durelli, pela disponibilidade em participar desse trabalho.

A Universidade Federal de Minas Gerais, pelo suporte acadêmico durante a graduação em Comunicação Social e agora no mestrado em Ciências da Computação.

Aos professores e à secretaria do PPGCC, pelas disciplinas, pelo apoio administrativo e por manter a excelência do Programa de Pós-Graduação em Ciências da Computação em meio aos desafios impostos pela pandemia e pela necessidade do ensino remoto.

Ao grupo de pesquisa ASSERG, pelo conhecimento compartilhado

Aos colegas de trabalho e à direção da empresa, por estimular a realização do Mestrado e por contribuir para a realização da pesquisa.

"*Somewhere, something incredible is waiting to be known.*"
(Sharon Begley)

Resumo

JavaScript é a linguagem de programação mais utilizada em aplicações web. Todos anos, uma nova versão dessa linguagem é lançada, adicionando novas funcionalidades muitas vezes mais performáticas e seguras. Entretanto, nem todos navegadores (e suas respectivas versões) são compatíveis com esses lançamentos. Da mesma forma, a adoção dessas funcionalidades pelos desenvolvedores não se dá de maneira imediata. Nesta dissertação, nós inicialmente aplicamos um questionário com 54 desenvolvedores JavaScript (com pelo menos 5 anos de experiência), obtendo uma taxa de resposta de 72%. O objetivo é compreender quais são os principais motivos e desafios na adoção de novas funcionalidades JavaScript. A partir dos resultados, percebemos que questões ligadas à qualidade de código, como legibilidade, manutenibilidade e velocidade de desenvolvimento são os principais motivos para a adoção dessas novas funcionalidades. Os resultados também mostraram a importância dos transpiladores JavaScript, ferramentas que transformam o código da nova funcionalidade em um código JavaScript com sintaxe mais antiga, para superar o desafio da compatibilidade com os navegadores. Em seguida, realizamos um estudo de mineração em projetos open-source JavaScript para verificar o uso do plugin Babel, que é o transpilador JavaScript mais utilizado. Detectamos a presença do Babel em 35% dos top 1000 projetos JavaScript do GitHub. Também investigamos, por meio de uma mineração de dados no Stack Overflow, os desafios enfrentados para utilizar essas novas funcionalidades. Concluímos que os transpiladores desempenham um papel importante no desenvolvimento de software moderno. Sem a presença dessas ferramentas, os desenvolvedores seriam limitados no uso de novas funcionalidades JavaScript devido a incompatilidade destas com os navegadores e suas versões mais antigas.

Palavras-chave: JavaScript, Transpiladores, Babel, Compatibilidade de Browsers, Manutenção de software

Abstract

JavaScript is the most used programming language in web applications. Every year, a new version of this language is released, adding features that are often more performant and secure. However, not all browsers (and its respective versions) are compatible with these releases. Likewise, the adoption of these functionalities by developers does not happen immediately. In this master thesis, we initially applied a questionnaire to 54 JavaScript developers (with at least five years of experience), obtaining a response rate of 72%. The goal is to understand the main reasons and challenges when adopting new JavaScript features. From the results, we realized that motives related to code quality, such as readability, maintainability, and development speed, are the main reasons for adopting these new features. The results also showed the importance of JavaScript transpilers, tools that transform new functionality code into JavaScript code with older syntax, to overcome the challenge of compatibility with browsers. Then, we do a mining study on open-source JavaScript projects to verify the use of the Babel plugin, which is the most used JavaScript transpiler. We detected the presence of Babel in 35% of GitHub's top-1k JavaScript projects. Through data mining on Stack Overflow, we also investigated the challenges when using these new features. We conclude that transpilers play a relevant role in modern software development. Without these tools, developers would have limited usage of new JavaScript features due to their incompatibility with browsers and their older versions.

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

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

JavaScript is one of the most important languages in modern software development, accordingly to a survey conducted by Stack Overflow [33]. The language is also listed as one of the most wanted programming skills in the job market [12]. The term *JavaScript* appeared in more than 50k job openings listed on the Indeed platform, which is the biggest job listing site in the world.[1]

Like most programming languages, JavaScript constantly evolves, adding different features in each new release. The language follows the ECMAScript specification [36], which has a process to define when a new implementation (*proposal*) is ready to become part of the ECMAScript. Since 2015, a new ECMAScript version has been released each year, introducing new JavaScript features. In 2020, for example, ECMA released the ECMAScript 2020, introducing features like Optional Chaining and Nullish Coalescing Operator.[2]

Unfortunately, when a new ECMAScript version is available, it does not mean that it can be largely used. Unlike backend programming languages, like Java and Python, JavaScript is a client-side language that relies on a browser to work. Since most browsers are held and maintained by private corporations, it depends on the company's decision to make the browser compatible with the new release. For example, many banking applications rely on Internet Explorer to run. Thus, it is a challenge to move to modern browsers for some businesses.[3] To mitigate this problem, developers often use JavaScript transpilers (like the Babel transpiler [3]) to generate a source code that will be retro-compatible with old browser versions. Although the developer community recognizes JavaScript as a very important language and new JavasScript features are available yearly, only some studies investigate why and how developers adopt them in their projects [14, 24]. Therefore, more information about the motives that lead programmers to use a new JavaScript feature

[1] https://www.thebalancecareers.com/top-best-job-websites-2064080
[2] https://exploringjs.com/impatient-js/ch_new-javascript-features.html
[3] https://dzone.com/articles/major-cross-browser-compatibility-issues-faced-by

is needed. We also need more evidence to clarify if browser compatibility is a barrier to using these new releases.

1.2 Proposed Work

1.2.1 A Survey Study

First, in this master thesis, we surveyed 54 experienced JavaScript developers to identify the reasons that led them to adopt new JavaScript releases. The participants work for well-known companies, such as Apple, eBay, Google, Delivery Hero, Walmart, Inter Bank, and others. We design a survey with 8 questions (mainly open-ended). We achieved a 72% of answer rate on the survey (39 interviewers filled out the form).

We also aim determine if browser compatibility issues are a concern when adopting these latest features. Since not all browsers are compatible with new JavaScript releases, the assumption was that cross-browser issues could be a limitation. Developers may not be using new JavaScript features to avoid problems with older browsers. In the survey, we use two questions to shed light in this matter. We also identify if developers have ever faced cross-browser issues when using a new JavaScript feature.

Finally, we use the survey to identify if developers leverage tools and frameworks to avoid those compatibility issues. With an open-ended question, we ask if the participants use any transpiler, framework, JavaScript supersets, and others; in their projects. The rationale behind this question is to identify if these tools help developers to deal with cross-browser issues when adopting new JavaScript Features.

1.2.2 A Mining Study

Based on the findings of this survey, we conduct a second study to explore one of the most adopted development tools in JavaScript: the Babel transpiler [3]. For this purpose, we design and develop a data mining study to assess the adoption and challenges of new JavaScript features based on the usage of the Babel transpiler. This study explores three data sources: GitHub, Stack Overflow, and *Can I Use?* platform [9]. We

use GitHub as a data source to have a significant amount of projects to analyze. We investigate the top-1k JavaScript projects ranked by the number of stars. Then, we use Stack Overflow data to understand if developers struggle to use new JavaScript features. The assumption is that if developers are troubled by new features, they could assess Stack Overflow community to overcome these challenges. Finally, we use *Can I Use?* data to access the browser compatibility of the latest JavaScript releases. The platform provides data about well-known browsers (like Chrome, Safari, Edge, Firefox, and others) and their respective versions that are compatible with a given front-end feature. The platform provides information regarding HTML tags, CSS properties, SVG properties, or JavaScript features. The focus of the second study is to identify the average compatibility of new JavaScript features among well-known browsers. In summary, we perform the following tasks:

1. First, we assess the occurrences of Babel plugins in the top-1k GitHub JavaScript projects to verify the usage of plugins related to new JavaScript features.
2. Second, we search for questions on Stack Overflow related to the usage of Babel plugins and new JavaScript features.
3. Finally, we parse the *Can I Use* platform data to identify the compatibility of new JavaScript features among the most used browsers.

1.3 Results

Based on our studies, we provide the following major findings:

- **Code quality is the main reason to adopt new JavaScript features.** About 84% of the participants mentioned at least one of the code quality-related characteristics, like maintainability, readability or simpler syntax.
- **Browser compatibility issue is not a concern among developers.** Only 20% of all participants mentioned that they faced browser compatibility issues when using a new JavaScript feature.
- **Babel transpiler has a relevant presence in the JavaScript ecosystem.** 35% of the studied JavaScript projects rely on Babel plugins.
- **Developers often struggles when using Babel transpilers.** Most of the Stack Overflow questions were on the development phase and mainly related to how properly import Babel plugins.

- **New JavaScript feature has, on average, 86% of compatibility with browsers**. It indicates that almost 14% of the web traffic could face a JavaScript error if developers do not use tools to make it compatible with older browser versions.

1.4 Publication

The second study was submitted to the *IEEE Software* journal and it is now under major review.

1.5 Outline of the Thesis

The remaining of this master's thesis is divided in the following chapters:

- **Chapter 2** details related work and introduces the main concepts involving JavaScript, cross-browser issues, JavaScript transpilers, and the Babel transpiler.
- **Chapter 3** presents the survey conducted with 54 experienced JavaScript Developers (with 72% of answer rate) regarding the adoption of new JavaScript features and how cross-browser issues can impact this adoption.
- **Chapter 4** presents a data mining study, discussing the usage of a very common JavaScript transpiler, Babel. We investigate open-sources projects that uses this tool, and the challenges on the adoption of new JavaScript features.
- **Chapter 5** concludes this master thesis and presents the final remarks, limitations, and future work.

Chapter 2

Background and Related Work

In this chapter, we present the technical background needed to understand our work. First, in Section 2.1, we discuss the concept of cross-browser issues. Then, in Section 2.2, we introduce new JavaScript features by presenting the TC39 process. In Section 2.3, we briefly explain JavaScript transpilers and the Babel transpiler, which is an important tool and a major subject of Chapter 4. In Section 2.4 we detail the related work. Finally, in Section 2.5 we present final remarks of this chapter.

2.1 Cross-Browser Issues

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, there are seven other browsers responsible for 16% of the Web traffic.[1] This profusion of options for Web browsing shows that the user has a lot of flexibility [19], 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 [20]. Many of these issues are perceived by different look and feel of Web apps running on different browsers.

Nowadays, CBC issues in the JavaScript era can impact the site functionality or make the application not work at all. This scenario is classified as a behavior cross-browser incompatibility (XBI I), which prevents the users from accessing part of the application's functionality [28]. These XBI I issues are a major concern for Web developers, because they affect the site's reliability and performance, which can impact the revenue of a company. Accordingly to Akamai,[2] a global content delivery network, a 100-millisecond delay in website load time can hurt conversion rates by 7%, for example.

These CBC issues are a concern when developers want to use the latest features of

[1] https://gs.statcounter.com/browser-market-share
[2] https://tinyurl.com/23wf9t5s

JavaScript [21]. Due to the large number of browsers and operational systems, when new JavaScript features are available, not all browsers are compatible with them. It may lead developers to postpone the adoption of new features (JavaScript proposals [37]), since not all browsers will be able to interpret them, possibly causing bugs when the user lands into the application.

2.2 New JavaScript Features in a Nutshell

To understand the usage of the new JavaScript features, we first need to explain how the language is maintained and how it evolves. JavaScript is a programming language that follows the ECMAScript specification, create by the ECMA International [13], an association that defines the standards for technologies. This institution has many committees, which are responsible for the standardization of a specific technology. In this scenario, we have the Technical Committee 39 (TC39), which is the committee designate to define ECMAScript specifications. TC39 is composed of JavaScript developers, academics, and members of big tech companies [36]. This committee is in charge of evaluating proposals for additional features in the ECMAScript standard.[3]

The idea of a proposal is adding specification regarding new features that improve the JavaScript development. To became a standard feature, the proposal has to pass trough five stages:

- **Stage 0**: A new feature is proposed and described.
- **Stage 1**: It makes the case for the addition, describes the solution, and identifies the potential challenges.
- **Stage 2**: It describes the syntax and semantics using formal specification language.
- **Stage 3**: It states that further refinement will need feedback from implementations and users.
- **Stage 4**: The feature is ready to be formally added to the ECMAScript standard.

When a feature is under stages three and four, browsers can implement their specification to make it available to developers. However, this process can take some time, and not all browsers implement the latest ECMAScript at the same pace. Also, not all users update their browser version regularly, which is another challenge of using these new

[3] https://tinyurl.com/4bf6yhab

features in Web development. To illustrate that, the *AsyncGenerator*,[4] which became a standard ECMAScript feature (Stage 4) in 2017, still has only 93% of compatibility across the most common browsers[9], which means that 7% of the global Web traffic will face cross-browser issues when landing in a site that uses this feature. Newer proposals, like *Logical AND assignment* (ES2021), has an even smaller compatibility rate, 88% [9].

To use these new features and avoid CBC issues, developers often rely on JavaScript transpilers. The next Section 2.3 presents a briefly explanation about this subject.

2.3 JavaScript Transpilers

2.3.1 Overview

A transpiler is defined as a set of tools that take the source code and, after the transpilation process, generates a source code written in another target language that is syntactically equivalent to the source one [2]. Most transpilers use Abstract Syntax Tree (AST) as an intermediary step when converting the source file into its final version [7]. The AST process breaks down the initial code, organizing it with meaningful metadata. Then, the source code is rewritten in the output format, which, in this case, is pure JavaScript code.

There are different types of JavaScript transpilers. Some of them, like Typescript, CoffeeScript, and Dart, have specific language specifications. After compilation, the respective code is transpiled into a JavaScript code [16]. Other types of transpilers are *source-to-source* tools [30], that means it transpile JavaScript code into an older version of JavaScript code. This process makes it possible to use JavaScript proposals even in the early stages. Since these plugins transpile the proposed feature into an old (and more compatible) JavaScript syntax, developers can leverage using new features without concerning with cross-browser issues. In the following section, we present Babel and a few examples of code transformation using some of its plugins.

[4] https://tinyurl.com/3ahmz33v

2.3.2 The Babel Transpiler

Babel is a well-known JavaScript transpiler used by the developer community. With around 45 million weekly downloads,[5] this toolchain is used to transform ECMAScript features (released after 2015) into a JavaScript code that is compatible with older browsers versions. It also adds polyfills to make some features available. Polyfill is a chunk of code that makes it possible to use modern functionality on outdated browsers.[6] JavaScript developers resort to this tool often, which is also present in famous JavaScript libraries, such as VueJS and React. Therefore, we can affirm that Babel is indirectly being used by many projects that use JavaScript libraries as a dependency, even if the project is not explicitly importing a Babel module. Babel is a *source-to-source* toolchain. Thus, it uses an AST in a three-step process.[7] First, it parses the source code into an AST. Then, it traverses the AST, adding, removing, and updating nodes in the three based on the rules defined by the Babel plugins. Finally, it takes the transformed AST to output a string of JavaScript code compatible with older browsers. In this step, Babel also generates a source map for debugging purposes.

For example, the plugin *plugin-proposal-class-properties* takes static declared fields and converts them into object properties. Figure 2.1, extracted from Babel documentation, presents the source code (written with static public methods, which became available in ES2015) and its output version. As we can see, the generated code is bigger and more complex. Luckily developers can rely on Babel source maps to debug code in the source version instead of debugging in the transpiled version. It is important to note that the developer can configure which range of browsers they want to cover. Therefore, the generated code can be smaller if Babel can use modern JavaScript syntax when transpiling a specific feature.

Another Babel plugin is *plugin-proposal-optional-chaining* about the Optional Chaining feature. This feature makes it possible to read the value of a nested property without having to check if each parent in the chain exists. It is a valuable feature when analyzing a chain of connected JavaScript objects [4]. Figure 2.2 presents an example (from Mozilla documentation [4]) regarding the usage of this feature. In lines 1 to 6, there is a declaration of a nested object, where the `adventurer` is in the first level, and `name` inside `cat` is the inner property of this object. In line 8, it tries to assign a non-existent property (`dog.name`) to a variable called `dogName`. The Optional Chaining feature (represented by the `?.` notation) makes the `dog` property optional, which means it may exist or not in the `adventurer` object. In the example, there is no `dog` object inside `adventurer`.

[5] https://www.npmjs.com/package/@babel/core
[6] https://developer.mozilla.org/en-US/docs/Glossary/Polyfill
[7] https://medium.com/front-end-weekly/a-world-of-javascript-transpilers-b3b7b880a1be

source

```
class Bork {
  static a = "foo";
  static b;

  x = "bar";
  y;
}
```

output

```
var Bork = function Bork() {
  babelHelpers.classCallCheck(this, Bork);
  Object.defineProperty(this, "x", {
    configurable: true,
    enumerable: true,
    writable: true,
    value: "bar",
  });
  Object.defineProperty(this, "y", {
    configurable: true,
    enumerable: true,
    writable: true,
    value: void 0,
  });
};

Object.defineProperty(Bork, "a", {
  configurable: true,
  enumerable: true,
  writable: true,
  value: "foo",
});
Object.defineProperty(Bork, "b", {
  configurable: true,
  enumerable: true,
  writable: true,
  value: void 0,
});
```

Figure 2.1: Example of Class Properties plugin

Consequently, an `undefined` value is assigned to no `name` property inside `dog` (as demonstrated in lines 9 and 10). Similarly, line 12 represents a tentative to access a non-existent method (`someNonExistentMethod`) in `adventurer` object. The result is an `undefined` being written in the console since the property does not exist in the main object. The Optional Chaining feature is essential to avoid a code break when there is no guarantee that the parent property will exist in the scope.

```
1  const adventurer = {
2    name: 'Alice',
3    cat: {
4      name: 'Dinah'
5    }
6  };
7
8  const dogName = adventurer.dog?.name;
9  console.log(dogName);
10 // expected output: undefined
11
12 console.log(adventurer.someNonExistentMethod?.());
13 // expected output: undefined
14
```

Figure 2.2: Example of Optional Chaining Babel from MDN

source

```
foo?.bar;
```

output

```
foo === null || foo === void 0 ? void 0 : foo.bar;
```

Figure 2.3: Example of Optional Chaining Babel plugin

Figure 2.3 presents another example from Babel documentation. The source code, with Optional Chaining feature, is converted into a chain of ternary operators. However, based on the size of the main object and its nested properties, the output ternary chain can be tough to read. Therefore, developers resort to Babel source maps when debugging issues with a transpiled source code version with Optional Chaining feature.

2.3.3 *Can I Use?* Platform

Can I use? [9] is an online platform that provides data regarding front-end technologies on desktop and mobile browsers.[8] Users can search for front-end features (like HTML tags, JavaScript functionality, or CSS properties), and the platform displays a table showing the feature compatibility among the most common web browsers (Chrome, Firefox, Safari, Edge, IE, and others). The platform also shows data regarding the global usage of each browser, utilizing data from *Statcounter* application.[9] We use data extracted from this platform to assess browser compatibility in the second study.

Developers use this platform to verify if they can adopt a feature in their projects. Programmers can adopt or postpone the adoption of a front-end feature based on the application's target audience.

Figure 2.4 presents an example of *Can I Use?* table for the Optional Chaining feature. On the left part of the figure, it is possible to check an estimate (93.96%) of the global usage. It means that almost 94% of the global web traffic is using a browser compatible with the Optional Chaining feature. It is also possible to see in the figure a breakdown of browser compatibility. Each column in the table displays compatibility with a specific browser. In the first column, it is possible to check that Chrome versions from 4-79 do not support Optional Chaining, thus they are displayed in red. In the table, it is also possible to see that Internet Explorer (IE) does not have versions compatible with this feature. The platform *Can I Use?* is an open-source application. However, not all front-end features have data available in this platform. Users can open an issue in *Can I Use?* GitHub repository to request the inclusion of a new feature. Based on the number of requests, maintainers will prioritize them.

2.4 Related Work

In this section, we provide an overview of the related work. Mesbah *et al.* [20] define how cross-browser issues impact Web software development. The authors proposed an automated method for cross-browser compatibility testing Web applications. The authors developed a tool to compare the rendered Web application across multiple browsers, comparing their similarities and differences. Their results show that cross-browser issues are a

[8] https://caniuse.com/ciu/about
[9] https://gs.statcounter.com/about

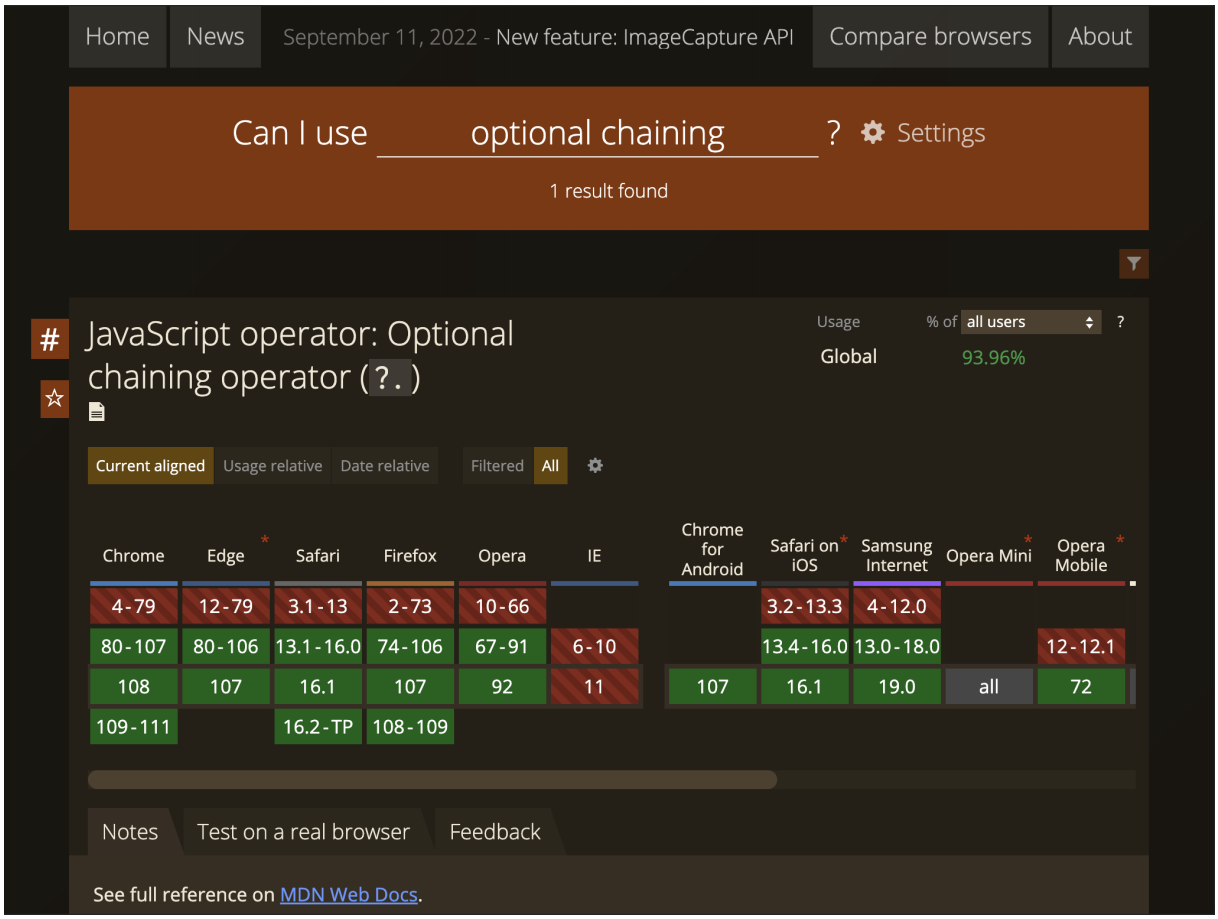

Figure 2.4: Optional Chaining compatibility - Can I Use Platform [9]

major problem in several open-source projects. The authors also propose a classification for different issues:

1. Behavior issues involve the difference in the behavior of the individual functional components within a page.

2. Structure issues refer to the difference in the layout of the page.

3. Content issue refers to the difference in the content of individual components of the Web page.

Guoquan Wu *et. al.* [39] proposed a technique to detect cross-browser issues for JavaScript-based Web applications. Their tool, *X-Check*, is a novel cross-browser testing technique tool based on record/replay. *X-Check* leverages Mugshot to implement record/replay functionality using standard JavaScript language, providing event capture/replay on unmodified browsers. Xu and Zeng [40] proposed a technique for statically analyzing cross-browser compatibility problems. The work aimed to detect whether Web applications contain HTML5 incompatible features, generating a report containing the HTML5 incompatible features in the Web site for developers. As an evolution of this

research, Xu *et. al.* [41] later proposed another tool called X-Diag, focused on the debugging of cross-browser issues. The tool narrows down the root causes of cross-browser issues step-by-step by checking whether such issues are caused by incompatible DOM APIs, CSS properties, or Html elements.

Choudhary *et. al* [27] proposed another tool, called WEBDIFF, focused on the behavior of a Web application in different Web browsers. The tool works identifying differences in behavior as potential issues and reports them to the developers. Given a page to be analyzed, the comparison is performed by combining a structural analysis of the information in the page's DOM and a visual analysis of the page's appearance, obtained through screen captures. These related works demonstrate that cross-browser compatibility issues are a problem in modern software development, and how researchers continue to develop tools to identify and mitigate them.

Other studies are focused on the JavaScript development challenges and their lack of standardization, Kyriakou *et. al* [17] presented the incompatibility and complexity problems in Web development when using JavaScript. The authors also proposed a tool to harmonize JavaScript-oriented Web development and web standards of the ECMAScript 6 "Harmony" specification.

To address JavaScript Web standards Lee *et. al* [18] presented a formal specification and implementation of SAFE, a scalable analysis framework for *ECMAScript*, developed for the JavaScript research community. This was an attempt to provide both formal specifications and its open-source implementation for JavaScript. The tool presented a framework that supported a level of intermediate representation: an abstract syntax three similar to the JavaScript source code. This approach is also fundamental to JavaScript transpiler, such as *Babel*. Regarding transpilers, Kimura *et. al* [16] 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. Through the study, the authors demonstrate how a transpiler can significantly reduce development time and also simplify source code with huge effects compared with other existing tools. Although the *Escapin* is a transpiler for specific usage, this study demonstrates how developers can leverage transpilation tools to improve software development. In the same way, *Babel* (and other JavaScript transpilers) has an important role in Web development.

Reiser *et. al* [26] proposed another JavaScript compiler called *Speed.js*. This tool compiles JavaScript/Typescript code into WebAssembly, a new standard for native execution supported by all major browsers. Although this tool is classified as a compiler, not a transpiler (because the final code is in a different language than the source one), the study also demonstrates how tools can use an AST to generate a better, compatible, and performative code across the Web.

Sayed *et al.* [30] proposed a JavaScript transpiler, which is an instrumented version

with the flow-sensitive security monitor inlined. The tool is focused on the security of the transpiled code since JavaScript projects often use many open-source libraries that cannot guarantee security. Thus, the authors present and discuss the implementation of the inlining transpiler and assess empirically its security effectiveness. Finally, JavaScript has been the focus of software engineering research for a long period [6, 10, 15, 22, 23, 31, 32]. Similarly, the Stack Overflow dataset is an important source of data for researchers nowadays [1, 25, 29, 34, 35, 38].

As far as we know, the related literature tackles different browser compatibility issues and proposes distinct tools to handle this issue. Other studies also describe how transpilers work in general, with different applicability and in distinct programming languages. To the best of our knowledge, none of them focused on the new JavaScript features and their impact on cross-browser issues, which was the main focus of this work.

2.5 Final Remarks

This chapter briefly presented the technical background related to this master thesis. Specifically, we detailed the concepts of Cross-browser issues, new JavaScript features, and JavaScript transpilers. We also detailed the motivation for understanding the new JavaScript feature's usage and challenges and the related work.

Chapter 3

Survey: the usage of new JavaScript features

In this chapter, we present the survey conducted with experienced JavaScript developers. First, we present the study design in Section 3.1. Then, in Section 3.2, we present the main results of this study in two parts: (i) the main reasons for developers adopting new JavaScript features and (ii) the challenges cited by the participants. In Section 3.3, we discuss the implications of this survey study. In Section 3.4, we discuss the implications threats of validity. Finally, in Section 3.5 we present the final remarks of this chapter.

3.1 Study Design

We designed a survey study to understand what drives developers to adopt new JavaScript features into their projects. The following sections describe each part of this study.

3.1.1 Survey Questions

Although we find multiple sources on blogs and dedicated software engineering websites about the benefits of each new JavaScript feature, we did not identify studies about how and why developers start to use them. In this study, we define a new JavaScript feature as any TC39 JavaScript proposal [36], which means that the feature is being built but, is still not ready to become part of an ECMAScript release. We intend to understand what are the motivations that lead developers to adopt these experimental features. Thus, we created a form with eight questions, three closed-ended and five open-ended to shed

light on this subject. The questions and their goals are summarized in Table 3.1.

Table 3.1: Summary of survey questions about the use of new JavaScript features

#	Question Statements	Type	Options	Rationale
	Section I - About the participant			
1	Do you consider yourself a:	Closed-ended	Backend Developer Frontend Developer Fullstack Developer	To guarantee that participants were software developers.
2	Regarding the project(s) that you're currently working on, is it (are they) based in which country??	Closed-ended	<List with 258 country names>	To guarantee that participants are currently working on at least one project and, to have more geographical diversity on the answers.
	Section II - New JavaScript Features			
3	What motivates you to adopt a new JavaScript feature?	Open-ended		Understand the main reasons that lead developer to use new features of a new JavaScript release.
4	Do you face any issues/challenges regarding browser compatibility when using new JavaScript features?	Closed-ended	Yes No	Understand if browser compatibility is a challenge for adopting a new JavaScript feature. If No, it goes to Section IV.
	Section III - Compatibility Challenges/Issues			
5	What compatibility challenges/issues do you face when using the new JavaScript features?	Open-ended		Understand the most common browser compatibility issues when using new JavaScript features.
6	How do you overcome these compatibility challenges/issues?	Open-ended		Understand the most common resolutions when facing browser compatibility issue.
	Section IV - Tools and Other Issues			
7	Do you use any transpilers (like Babel), JavaScript supersets (TypeScript/CoffeScript), or any other JavaScript framework/tool to avoid browser compatibility issues? If yes, which one(s) and why?	Open-ended		Understand if developers leverages tools/frameworks to avoid browser issues when using the latest JavaScript features.
8	Do you face any other challenges/issues when using new JavaScript features? If yes, please comment below.	Open-ended		Understand if there are common issues besides browser compatibility when using new JavaScript features.

The questions of the first section of the survey (questions one and two) are required but were not subject of this study. They were added to guarantee that the participants were developers and current involved in at least one software project. In Section II (questions three and four), both are required. Question three is open ended to get as many as possible motivations from the developers. Question four is required, but a simple Yes/No answer about browser compatibility issues. In case participant replied Yes, he/she is directed to Section III (questions four and five, both required). This section presented specific questions about browser compatibility issues when using new JavaScript features. In case the participant replied No on question four, it skips Section III and directed section IV (questions six and seven). These questions are related to tools to help the new JavaScript features adoption and if there are other issues when using them. It is important to reinforce that, if the participant landed into Section III they are also is directed to Section IV, to answer questions six and seven as well.

3.1.2 Participants

We submitted the form to 54 developers and we received an answer rate of 72% (39 answers). This high rate is explained by our approach when asking for participants: the author directly contacted all developers. They were all developers with at least five years of experience and worked or are currently working in the same outsourcing company as the author. This outsourcing company has approximately a thousand and two hundred employees, of which about eight hundred of them are developers. Since it is a multinational outsourcing company, it allocates developers to different clients/companies. That is beneficial because it allows us to have different tech stacks and backgrounds in the same company. We contacted the developers through tools like Linkedin, Whatsapp, and Google Chats. The participants were mainly developers that work on large projects for companies like Walmart, Apple, Google, eBay, Inter Bank, Delivery Hero, and Takeaway (some working throughout this outsourcing company, others currently employed by them). The author actively reminded them to answer in one-month time spam.

3.1.3 Data Classification

Since we have five open ended questions, we categorized and grouped similar terms into same answer category. Thus, we were able to group synonyms into meaningful categories. We rely on *thematic analysis* [11] to group answers or parts of answer in a specific category. We identified and record themes in textual documents, using the following steps: (1) initial reading of the answers, (2) generating a first code for each answer and its parts, (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 thesis, while Steps 4 and 5 were done together by the author and the advisors until consensus was achieved.

3.2 Results

3.2.1 Reasons for adoption New JavaScript features

After the thematic analysis on the third question of the survey (**What motivates you to adopt a new JavaScript feature?**), we grouped the answers into nine categories (*Simpler Syntax, Maintainability, Readability, Development Speed, Code Performance, Project Rule/Standard, Browser Compatibility, Framework Rule/Standard, and Code security*. To expand our data analysis, we grouped these categories into three groups:

- **Code Quality** - Reasons related to the code quality and development experience: *Simpler Syntax*, *Maintainability*, *Readability* and *Development Speed*.
- **Language enhancements** - Reasons related to improvements in the programming language itself: *Code Performance* and *Code Security*.
- **Non-Technical** - Reasons that does not depends on the developer decision: *Project Rule/Standard* and *Framework Rule/Standard*.

Figure 3.1 summarizes the distribution of the reasons in these three major groups. Notice that code quality is the most prevalent group (57), followed by language enhancements (12) and non-technical (8) reasons.

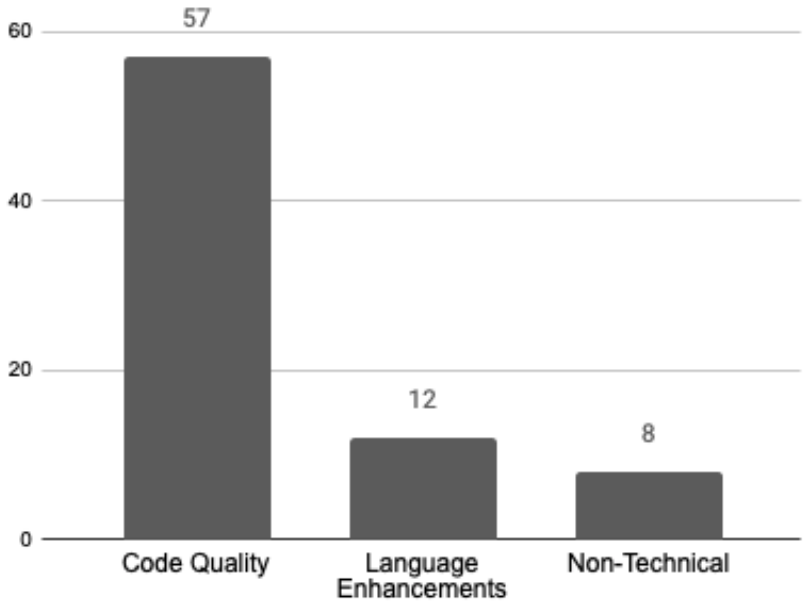

Figure 3.1: Reasons grouped by category

Table 3.2 presents the categories, theirs respective group, and the number of times that they were mentioned by the survey participants. The *Simpler Syntax* of a new JavaScript feature, compared to the syntax of an old implementation, is the most relevant reason for developers when adopting a new feature, cited by almost 54% of the participants. Other reasons related to code quality (like *Maintainability* and *Readability*) are also important, mentioned by 41% and 28%, respectively.

Table 3.2: Reasons on the adoption of new JavaScript Features

Motive	Group	# of Mentions
Simpler Syntax	Code Quality	21
Maintainability	Code Quality	16
Readability	Code Quality	11
Development Speed	Code Quality	9
Code Performance	Language enhancements	8
Project Rule/Standard	Non-Technical	3
Browser Compatibility	-	2
Framework Rule/Standard	Non-Technical	2
Code security	Language enhancements	1

On the other hand, characteristics related to the language enhancements, such as *Code performance* (the velocity that the browser will take to execute the block of code), were cited by only 23% of the participants. The *Code security* of a new implementation was mentioned just one time, representing 2% of the sample. Lastly, 12% of the participants mentioned that one of the reasons to use a new JavaScript feature is because of something outside his/her decision, like *Project Rule/Standard* or *Framework Rule/Standard*.

Notice that two participants mentioned *Browser Compatibility* as a reason. At a first glance, browser compatibility should not be considered as a reason for the adoption, but a blocker, since not all browsers are compatible when a new JavaScript release is available. These two participants are probably targeting only the latest browser versions in their projects or did not fully understand the question. Thus this category was not part of any group.

3.2.2 Challenges on the adoption of new JavaScript Features

3.2.2.1 Browser Compatibility

Only nine participants mentioned that they faced a browser compatibility issue when they tried to use a new JavaScript issue. This number represents 23% of the participants. Of this total, only two participants also mentioned that, along the browser compatibility issue, they also faced a Framework compatibility issue.

Regarding the results of the question four of the survey (**How do you overcome these compatibility challenges/issues?**), we find that around 67% of the participants used Polyfill,[1] which is a service to provide some functionality on older browsers that do not natively support it. Overall, 33% used transpilers and another 33% mentioned that created specific workaround to overcome the browser issue. Finally, one participant mentioned that they had issue with a specific version of the a framework, and they fixed it by updating the framework version.

Regarding the question number five of the survey (**Do you use any transpiler, JavaScript superset, or any other JavaScript framework/tools?**), Figure 3.2 shows that 80% of the participants mentioned the use of at least one framework or JavaScript tool in their projects. Around 74% cited TypeScript, (a JavaScript superset) and 67% cited Babel (a JavaScript transpiler). This widely spread usage of JavaScript tooling may explain why Browser Compatibility does not represent a major concern when using a new JavaScript feature. By design, these tools let the developer use a modern JavaScript implementation and then generates a code that will be compatible with older browser versions.

Overall, only 20% of all participants mentioned that they faced browser compatibility issues when using a new JavaScript feature. Out of this total, 22% abandoned or postponed the adoption of a new JavaScript feature due to browser compatibility issue. The large usage of JavaScript tools like transpilers and supersets (80% of the participants use) can explain this low concern with compatibility issue.

[1] https://polyfill.io/v3

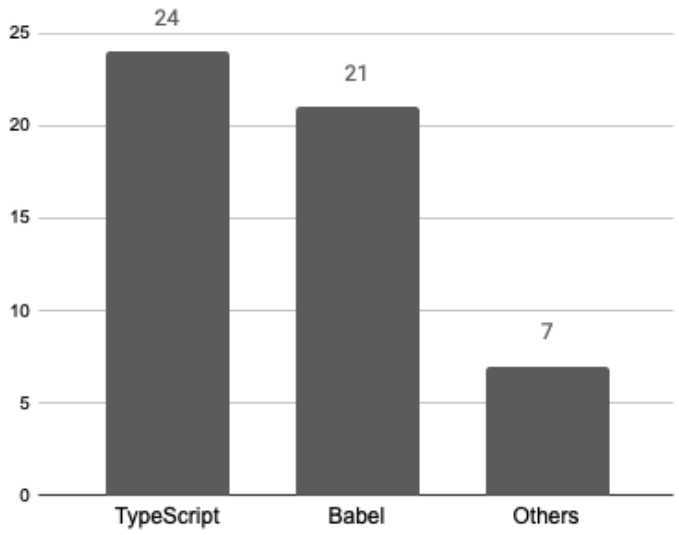

Figure 3.2: Use of JavaScript tools.

3.2.2.2 Other Challenges

In summary, 28% of the participants mentioned issues when using new JavaScript features. Of this total, the majority of them (34%) mentioned difficulties when configuring transpilers into their projects. Around 18% mentioned issues with the IDE support, indicating issues to use these new features in the work environment. Another issue mentioned by 18% of the participants who faced challenges, is the lack of support of the development team. It indicates that even when some developers are willing to adopt a new JavaScript feature, they may struggle to coordinate it with other developers working on the same project. In general, 18% mentioned compatibility issues with the JavaScript framework being in use in their project. Finally, only one participant mentioned problems with the code syntax of the new JavaScript feature. This low number may be explained because new releases often presents a simpler syntax, making the adoption easier.

Out of the 11 participants who cited other challenges/issues when adopting new JavaScript features, 36% of them mentioned issues with the configuration of the JavaScript transpilers, which is one of the tools responsible for the code syntax retro-compatibility.

3.3 Discussion and Implications

3.3.1 For researchers

Novel empirical data on the motivations on the adoption of new JavaScript features. Overall, developers decide to use new JavaScript features to improve their own development environment. The most common reason is the simpler syntax, cited by 54% of the participants. This can indicate that developers are looking for new JavaScript features to improve their productivity, since they can leverage the syntax simplicity and other code quality aspects of these new features.

Browser compatibility issues is not a huge concern when using a new JavaScript feature. Only eight participants mentioned that they had problems with browser compatibility issues when adopting a new JavaScript feature. This corresponds only 20% of the participants. Out of this sample, 67% mentioned that they used a technical workaround (Polyfill), and 33% mentioned the usage of JavaScript transpilers to overcome compatibility issues. This suggests that, even if a new feature is not compatible with a specific browser versions, developers can still uses it due to tools and transpilers that make the new code syntax available to older browsers.

3.3.2 For practitioners

Language enhancements like code security or code performance are not a main reason to adopt a new feature. Some new JavaScript features introduces code that executes faster in the browser and provides more security to the user (enabling encapsulation, private methods, etc; to JavaScript code). However, code enhancements characteristics does not encourage developers to adopt new JavaScript features. Only 23% participants mentioned these characteristics as a main reason to adopt a new feature. This can indicate that developers may not know these intrinsic characteristics of a JavaScript feature, thus they are not aware of the benefits of using them.

The usage of a JavaScript transpiler can be a challenge when adopting a new JavaScript feature. Although developers are not concerned about browser compatibility

issues, they may postpone the adoption of a new JavaScript release if they can not set up a transpiler properly. Out of the participants who mentioned another issues, 36% mentioned issues with transpilers plugins. This result can indicate that a absence of tool to transpile the new JavaScript code to a more compatible one can be a challenge on the adoption. That said, the concern with cross-browser issues is still present for a small number of developers. However, it only become a blocker when tools to overcome this challenge are not available.

3.4 Threats to Validity

In this study, we assessed the reasons that led developers to adopt new JavaScript features. By conducting a survey with thirty nine developers, we identify the motivations and challenges of this adoption. Although the survey participants are developers, working in projects for well-known companies such as Apple, Google, Walmart, eBay Takeaway, Delivery Hero, etc, their opinions may not represent the whole JavaScript developer community. Since all participants are currently working for privately held companies, their opinion can be biased by the company's technology standards. The survey questions may be interpreted in a different way for each participant, due to ambiguity or miss interpretation. To overcome this thread, we conducted a pilot survey. Based on the feedback of the pilot survey participants, we enhanced the questions wording and added a survey introduction explaining what we defined as a new JavaScript feature with examples. Finally, regarding the categorization of the open ended questions of the survey, it is subjected to human bias in the classification. To minimize this threat, we did a thematic analysis with the participation of the author and advisors, until consensus was achieved.

3.5 Final Remarks

In this section, we presented an empirical study on the motivation for the usage of new JavaScript features and the relation of cross-browser issues with this adoption. Through a survey with experienced developers, we took a look at JavaScript's new releases and theirs browser compatibility. We identified that, due to the use of tools like transpilers and JavaScript supersets, browser compatibility issues are not a huge concern among

developers that intend to use new JavaScript releases in their projects. We also identified that the use of these new features is related to the developer experience itself. That is, characteristics like code readability and coding velocity are more important than code performance or security.

Chapter 4

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

In this chapter, we present the data mining study regarding the usage of Babel plugin among GitHub open-source projects. First, we present the study design in Section 4.1. Then, in Section 3.2, we present the main results of this study in three parts: (i) the usage of Babel plugins, (ii) the issues found on Stack Overflow regarding the usage of proposals plugins, and (iii) the overall browser compatibility of JavaScript proposals. In Section 4.3, we discuss the implications of this survey study. In Section 4.4, we discuss the implications threats of validity. Finally, in Section 4.5 we present the final remarks of this chapter.

4.1 Study Design

Figure 4.1 presents an overview of the study design. We first analyse the top-1K ranked GitHub JavaScript projects and their usage of new JavaScript features. Then, we analyze Stack Overflow questions related to new JavaScript features. Finally, we assess the Can I Use? platform data to identify the browser compatibility of the new JavaScript features. The following subsections describe in details each one of these analysis.

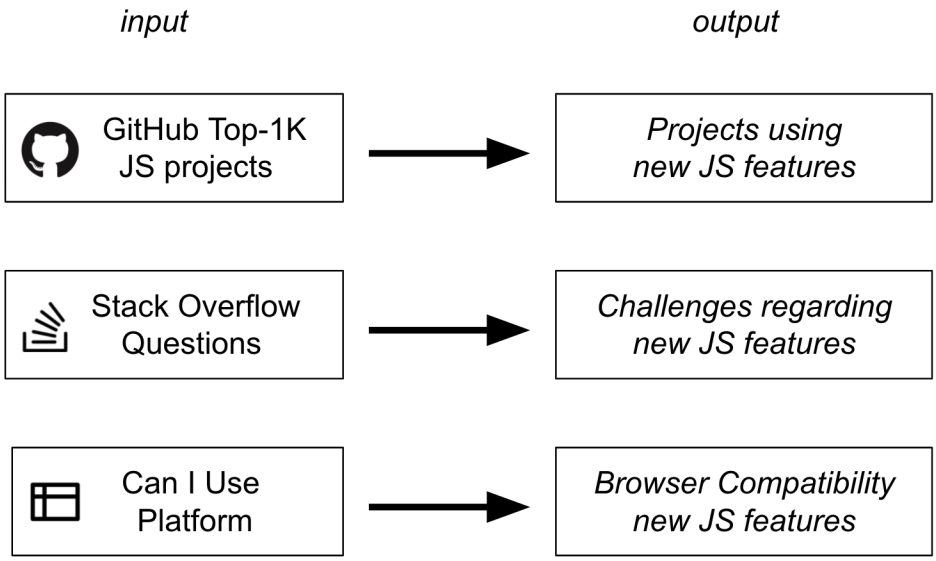

Figure 4.1: Overview of the study design.

4.1.1 Mining projects that rely on Babel

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 [5]. We rely on the GitHub search API,[1] 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.[2] 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[3] 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. This tool can identify plugins that are imported but are not used by the application. We found that 98% of the projects that imports Babel modules are indeed using them.

[1] https://tinyurl.com/34mkekhu
[2] https://babeljs.io/docs/en/
[3] https://github.com/depcheck/depcheck

4.1.2 Exploring StackOverflow questions

Next, we rely on the Stack Overflow Search API[4] 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.[5] 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* [11], 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 author of the thesis, while steps 4 and 5 were done together by all the author and the advisors until consensus was achieved. Thematic analysis is also used in the following two steps.

4.1.2.1 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, or how to run the project locally (*e.g., "Trouble Installing babel's plugin-proposal-export-default-from"*).
- **Development:** When the question is related to the plugin usage itself, with code

[4] https://api.stackexchange.com
[5] https://babeljs.io/docs/en/plugins-list

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"*).

Based on this classification, we aim to identify in which phase of development the transpiler usage related to the new JavaScript feature is more challenging for the developers.

4.1.2.2 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 a IDE / Text 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, functionality, and how to proper integrate the plugin within the code (*e.g., Babel plugin-proposal-decorators not working as expected"*).

4.1.3 Assessing browser compatibility

In this last analysis, we rely on the [9] platform 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 [9]. 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* [8], which became a JavaScript feature in the ES6 release (2015), has an estimate of 96.11% of global compatibility. This platform also indicates that the feature became available in Chrome Desktop in version 45. Although the platform already provides 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. This script examined the site data counting the number of browser versions available. Then, the script checked the compatibility of each version with a new JavaScript feature. With this, we are 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 [9] platform. This compatibility rate is an estimation between the global internet traffic and the browser/version usage.

4.2 Results

4.2.1 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 10 most common Babel modules found in these 345 projects are presented in Table 4.1. The *core* module is the top one, with 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.[6]

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

Table 4.1: Frequency of Babel modules

Module Name	#Frequency
core	282
preset-env	211
eslint	159
loader	141
cli	116
preset-react	59
plugin-transform-runtime	57
jest	56
plugin-proposal-class-properties	54

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 4.2, 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 (represented by the ... notation) 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.

In 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.

4.2.2 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. Figure 4.2 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)

Table 4.2: 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

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.

4.2.3 Browser compatibility

We found data for 18 out of the 30 available Babel proposal plugins. Since *Can I use?* [9] 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 4.3 presents the results by desktop versions, mobile versions, and the total of

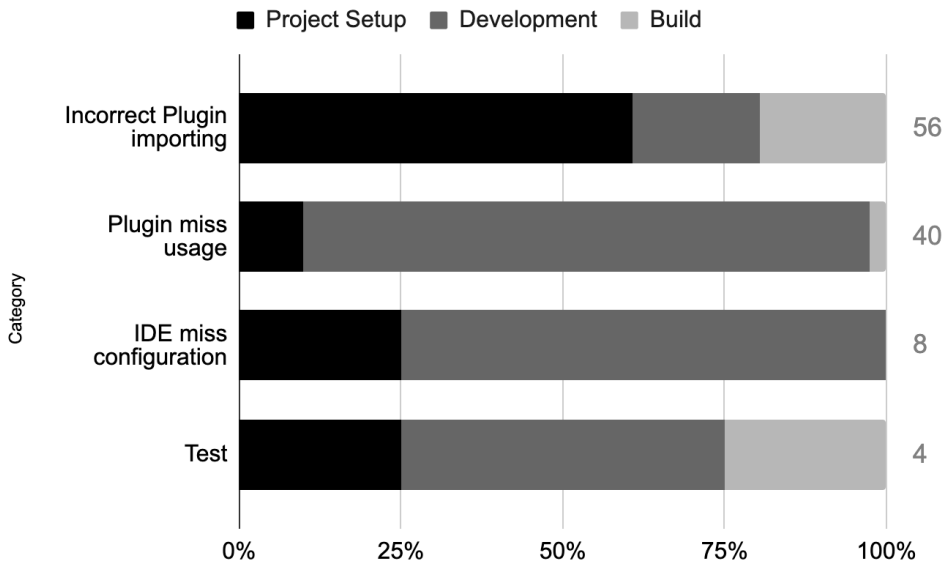

Figure 4.2: Stack Overflow Questions by topic and phase.

Table 4.3: 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**

compatibility rate of a given proposal plugin. Overall, we found out 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 (*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.

4.3 Discussion and Implications

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

4.3.1 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. 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. To reinforce, according to NPM data, React has around 45 million weekly downloads.[7] 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. 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.

4.3.2 For Researchers

Factors that influence the adoption of a new JavaScript feature. 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

[7] https://www.npmjs.com/package/@babel/core

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. Novel studies can focus on identifying the benefits 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.

4.4 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 study. Regarding the categorization of the Stack Overflow questions, it is subjected to human bias in the classification. To overcome this possible threat, we relied on thematic analysis [11]. 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.

4.5 Final Remarks

In this chapter, we presented an empirical study on using new JavaScript features and their impacts on browser compatibility. We looked deeper at a JavaScript transpiler and how it allows a developer to use new JavaScript releases even though most browsers do not yet support these. In future work, we plan to conduct another mining study to address the usage of new JavaScript features through other transpilers, like Typescript and Webpack. Since the usage of these tools is increasing, it is important to investigate how they can contribute to modern software development. We also plan to conduct a study to

identify *how* repository maintainers' determines to introduce JavaScript proposal into an open-source project and *why* they choose to adopt a specific proposal instead of another. Lastly, we aim to understand how developers can use new JavaScript features in a faster and more widely spread way.

Chapter 5

Conclusion

In this chapter, we present the conclusion of this master thesis. First, in Section 5.1, we present the lessons and contributions provided by our work. Next, in Section 5.2 we present the limitations of the study. Finally, in Section 5.3, we present suggestions of future work.

5.1 Overview and Contributions

Our main contributions in this work are described next.

Reasons for adopting new JavaScript features. In Chapter 3, we presented a study with the main reasons that lead JavaScript developers to adopt new features in their projects. We concluded that factors related to code quality (maintainability, readability, velocity of development) are the main motives. We also found that cross-browser issues are not the primary concern among developers. Since most use transpilers, JavaScript supersets, and frameworks, these tools make it possible to adopt new JavaScript releases without worrying if the browser is compatible with the respective feature.

The relevance of Babel Tranpsiler. In Chapter 4, we developed a data mining study regarding adopting new JavaScript features by using Babel transpiler. We identify that this tool has a relevant presence in open-source projects (35% of the top-1k JavaScript open-source projects on GitHub uses at least one Babel plugin). Since well-known JavaScript frameworks (like VueJs and React) use Babel, we can affirm that many applications (that use these frameworks) also use Babel indirectly. After analyzing Stack Overflow questions, we shed light on some challenges developers faced when using Babel plugins. Ally, we identify that, with the Babel transpiler, developers can adopt new JavaScript features due to the lack of compatibility among the most common web browsers.

5.2 Limitations

The work presented in this master thesis has the following limitations:

- The survey is subjected to human bias and cultural factors. Since all the participants were related to the author, the survey sample does not reflect all the developer community opinion.
- Not all JavaScript transpilers were covered in the mining study. Since we focused on Babel, we may lose data regarding using new JavaScript features in projects that use another tranpilers.
- Not all JavaScript projects uses NPM as a dependency manager. Thus projects that uses other tools, like *Yarn* were not covered in the mining study. Since we focused on NPM, we may lose data regarding using new JavaScript features in projects that use another package manager.
- The existence of a proposal plugin in the open-source project is a good indication that this project is using a new JavaScript feature. However, we can not affirm that all imported proposals are in use by the application.

5.3 Future Work

We propose the following future work:

- In the mining study, we focused on the Babel transpiler. A similar study can be conducted with other tools, like Typescript. Since the usage of Typescript is increasing, it would be interesting to verify how this superset can influence the adoption of new JavaScript features.
- To elaborate a data mining study to verify if the proposal feature is really being in use by the open source project, for example, by analyzing the code syntax.
- To conduct a study to verify the average time that a JavaScript proposal goes from stage 0 to stage 4 in the TC39 process, and when developers start to use them.
- To create a tool to suggest JavaScript code refactoring based on new feature releases. For example, this syntactic tool would offer new features over old code.

- Finally, to conduct a survey to understand why some new features are often more adopted than others.

Bibliography

[1] Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem, and Emad Shihab. Challenges in chatbot development: A study of stack overflow posts. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 174–185, New York, NY, USA, 2020. Association for Computing Machinery.

[2] Bastidas F. Andrés and María Pérez. Transpiler-based architecture for multi-platform web applications. In *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, pages 1–6, 2017.

[3] Babel. The compiler for next generation javascript. https://babeljs.io/, 2022. [Online; accessed August 2022].

[4] Babel Plugins. Optional chaining (?.) - javascript — mdn. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining, 2021. [Online; accessed August 2022].

[5] 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.

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

[7] Byby Dev. Javascript transpilers. https://byby.dev/js-transpilers, 2022. [Online; accessed June 2022].

[8] Can I Use? Arrow functions — can i use... support tables for html5, css3, etc. https://caniuse.com/arrow-functions, 2021. [Online; accessed August 2022].

[9] Can I Use? Support tables for html5, css3, etc. https://caniuse.com/, 2022. [Online; accessed October 2022].

[10] Moumena Chaqfeh, Muhammad Haseeb, Waleed Hashmi, Patrick Inshuti, Manesha Ramesh, Matteo Varvello, Fareed Zaffar, Lakshmi Subramanian, and Yasir Zaki. To block or not to block: Accelerating mobile web pages on-the-fly through javascript classification, 2021.

[11] 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.

[12] Developer News. These programming languages were most in-demand in 2021. https://www.developer-tech.com/news/2022/mar/08/these-programming-languages-were-most-in-demand-in-2021/, 2021. [Online; accessed June 2022].

[13] ECMA International. Industry association for standardizing information and communication systems. https://www.ecma-international.org, 2022. [Online; accessed October 2022].

[14] Munawar Hafiz, Samir Hasan, Zachary King, and Allen Wirfs-Brock. Growing a language: An empirical study on how (and why) developers use some recently-introduced and/or recently-evolving javascript features. *Journal of Systems and Software*, 121:191–208, 2016.

[15] A. Javan Jafari, D. Costa, R. Abdalkareem, E. Shihab, and N. Tsantalis. Dependency smells in javascript projects. *IEEE Transactions on Software Engineering*, pages 1–1, aug 2021.

[16] 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.

[17] K. D. Kyriakou, I. K. Chaniotis, and N. D. Tselikas. The gpm meta-transcompiler: Harmonizing javascript-oriented web development with the upcoming ecmascript 6 "harmony" specification. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pages 176–181, 2015.

[18] Hongki Lee, Sooncheol Won, Joonho Jin, Junhee Cho, and Sukyoung Ryu. Safe: Formal specification and implementation of a scalable analysis framework for ecmascript. 2012.

[19] 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.

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

[21] Mozilla. Handling common javascript problems. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/JavaScript, 2022. [Online; accessed October 2022].

[22] R. Nascimento, E. Figueiredo, and A. Hora. Javascript api deprecation landscape: A survey and mining study. *IEEE Software*, 39(03):96–105, may 2022.

[23] 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.

[24] Amantia Pano, Daniel Graziotin, and Pekka Abrahamsson. Factors and actors leading to the adoption of a javascript framework. *Empirical Software Engineering*, 23(6):3503–3534, 2018.

[25] Soyeon Park, Wen Xu, Insu Yun, Daehee Jang, and Taesoo Kim. Fuzzing javascript engines with aspect-preserving mutation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1629–1642, 2020.

[26] Micha Reiser and Luc Bläser. Accelerate javascript applications by cross-compiling to webassembly. In *Proceedings of the 9th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages*, VMIL 2017, page 10–17, New York, NY, USA, 2017. Association for Computing Machinery.

[27] S. Roy Choudhary, H. Versee, and A. Orso. Webdiff: Automated identification of cross-browser issues in web applications. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10, 2010.

[28] Shauvik Roy Choudhary, Mukul R. Prasad, and Alessandro Orso. X-pert: Accurate identification of cross-browser issues in web applications. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, page 702–711. IEEE Press, 2013.

[29] 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.

[30] Bassam Sayed, Issa Traoré, and Amany Abdelhalim. If-transpiler: Inlining of hybrid flow-sensitive security monitor for javascript. *Computers & Security*, 75:92–117, 2018.

[31] Fabio Silva, Hudson Borges, and Marco Tulio Valente. On the (un-)adoption of JavaScript front-end frameworks. *Software: Practice and Experience*, 1:1–27, 2021.

[32] Leonardo Humberto Silva, Marco Tulio Valente, Alexandre Bergel, Nicolas Anquetil, and Anne Etien. Identifying classes in legacy JavaScript code. *Journal of Software: Evolution and Process*, 29(8):1–20, 2017.

[33] Stack Overflow. Stack overflow developer survey 2020. https://stackoverflow.com/, 2021. [Online; accessed June 2022].

[34] Mohammad Tahaei, Kami Vaniea, and Naomi Saphra. Understanding privacy-related questions on stack overflow. CHI '20, page 1–14, New York, NY, USA, 2020. Association for Computing Machinery.

[35] Youshuai Tan, Sijie Xu, Zhaowei Wang, Tao Zhang, Zhou Xu, and Xiapu Luo. Bug severity prediction using question-and-answer pairs from stack overflow. *Journal of Systems and Software*, 165:110567, 2020.

[36] TC39 - Committee. Specifying javascript. https://tc39.es/, 2021. [Online; accessed October 2022].

[37] TC39 - Committee. The tc39 process. https://tc39.es/process-document/, 2022. [Online; accessed October 2022].

[38] Liting Wang, Li Zhang, and Jing Jiang. Duplicate question detection with deep learning in stack overflow. *IEEE Access*, 8:25964–25975, 2020.

[39] 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.

[40] 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.

[41] S. Xu, C. Zhou, Z. Gu, G. Wu, W. Chen, and J. Wei. X-diag: Automated debugging cross-browser issues in web applications. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 66–73, 2018.