Assessing JavaScript API Deprecation

Romulo Nascimento[1], **Eduardo Figueiredo**[1] **(advisor), Andre Hora**[1] **(co-advisor)**

[1]Master's in Computer Science
Graduate Program in Computer Science (PPGCC)
Computer Science Department – Federal University of Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil
Admission: 08/2019 – Qualification: 10/2020 – Expected Final Presentation: 07/2021
{romulonascimento, figueiredo, andrehora}@dcc.ufmg.br

Abstract. *Building an application using third-party libraries is a common practice in software development. As any other software system, code libraries and their APIs evolve over time. In order to help version migration and ensure backward compatibility, a recommended practice during development is to deprecate API. Although studies have been conducted to investigate deprecation in some programming languages, such as Java and C#, there are no detailed studies on API deprecation in the JavaScript ecosystem. The goal of this master research work is to investigate deprecation of JavaScript APIs. In a first assessment, we analyzed popular software projects to identify API deprecation occurrences and classify them. We are now conducting a survey study with developers to understand their thoughts and experiences on JavaScript API deprecation. Lastly, we plan to develop a set of JavaScript API deprecation guidelines based on this master research result. Initial results suggest that the use of deprecation mechanisms in JavaScript packages is low. However, we were able to identify five different approaches that developers primarily use to deprecate APIs in the studied projects. Among these solutions, deprecation utility (i.e., any sort of function specially written to aid deprecation) and code comments are the most common practices in JavaScript. Finally, we found that the rate of helpful message is high: 67% of the deprecation occurrences have replacement messages to support developers when migrating APIs.*

Keywords: *API Deprecation, JavaScript, Software Library*

CBSoft Events: *SBES and SBLP*

1. Introduction

Building an application using third-party libraries is a common practice in software development. Libraries provide reusable functionality to client applications through their Application Programming Interfaces (API). API usage brings several advantages to a software development project [Tourwé and Mens 2003], such as cost and resources usage reduction. As a result, developers can focus on business core requirements and software quality may increase by relying on libraries that have been widely adopted, tested and documented [Moser and Nierstrasz 1996].

As any other software system, libraries and their APIs evolve over time [Granli et al. 2017]. Thus, functions and parameters might be renamed, updated, moved, or removed. Consequently, client applications need to migrate to the latest stable versions of their dependencies [Bogart et al. 2016]. To help version migration and ensure backward compatibility, a recommended practice in software development is to deprecate APIs. In other words, deprecation indicates that the use of a certain API should be avoided because it will be changed, removed or discontinued in a future version [Robbes et al. 2012].

Some of the most popular programming languages, such as Java and C#, provide native support mechanisms and tools to help developers explicitly deprecate their APIs [Sawant et al. 2018]. Indeed, recently, there have been many studies on deprecation practices and mechanisms mostly on those languages [Robbes et al. 2012, Bogart et al. 2016, Brito et al. 2018, Li et al. 2018, Sawant et al. 2019]. However, to the best of our knowledge, there are no detailed studies regarding API deprecation in the JavaScript ecosystem.

JavaScript has become popular over the last years [Santos et al. 2015]. According to a Stack Overflow Survey[1], JavaScript is the most popular programming language in this platform for the eighth consecutive year. GitHub also reports that JavaScript is the most popular language in terms of repository contributors[2]. Despite the growth on the use of JavaScript external libraries and APIs, little is known about JavaScript API deprecation mechanisms and practices. Unlike other popular programming languages, such as Java and C#, JavaScript does not provide native deprecation mechanisms.

This master research project aims at contributing to the software engineering community in the following ways:

- Understand deprecation in the JavaScript ecosystem
- Investigate how developers deprecate JavaScript APIs
- Understand how developers react do JavaScript API deprecation
- Develop a set of guidelines on JavaScript deprecation best practices

In a first assessment, we analyzed popular software projects to identify deprecation occurrences and classify them. We are now conducting a survey study with developers to understand their thoughts and experiences on JavaScript API deprecation. Lastly, we plan to develop a set of JavaScript API deprecation guidelines based on this work result. Initial results suggest that the use of deprecation mechanisms in JavaScript packages is low. However, we were able to identify five different approaches that developers primarily use to deprecate APIs in the studied projects. Among these solutions, deprecation

[1]https://insights.stackoverflow.com/survey/2020

[2]https://octoverse.github.com

utility (i.e., any sort of function specially written to aid deprecation) and code comments are the most common practices in JavaScript. Finally, we found that the rate of helpful message is high: 67% of the deprecation occurrences have replacement messages to support developers when migrating APIs [Nascimento et al. 2020].

2. Related Work

Sawant et al. (2019) conducted several studies to investigate Java API deprecation practices. The authors assessed the impacts, the needs, the reasons, and the patterns of API deprecation. They observed that the Java deprecation mechanism does not address all developers needs when it comes to deprecation. The authors also detected that Javadoc is not sufficient to understand the reasons behind deprecation occurrences; by mining other data sources such as source code, issue tracker data and commit history, they identified 12 reasons that trigger developers to deprecate APIs. They verified that most API client applications do not react to deprecation. Thus, they applied a survey to gather qualitative data from developers and try to explain this behavior [Sawant et al. 2019].

Robbes at al. (2012) studied deprecation in the context of the Smalltalk ecosystem. Brito et al. (2018) investigated the use of deprecation messages in Java and C#. The authors describe that 66.7% and 77.8% of Java and C# APIs, respectively, are deprecated with deprecation messages and that this rate does not evolve over time. Li et al. (2018) performed an exploratory study on Android API deprecation and identified that the Android framework is regularly cleaned-up from deprecated APIs and their maintainers ensure that deprecated APIs are commented to provide replacement messages. However, those APIs are not consistently annotated and documented and the existing documentation is not frequently updated [Li et al. 2018]. Many papers investigate how APIs evolve, measure breaking changes, and analyze their impact on client systems [Brito et al. 2019] [Xavier et al. 2017]. However, none them covers the JavaScript ecosystem.

3. Proposed Methodology

This master research work aims at understanding JavaScript API deprecation. In order to achieve its proposed goals and contributions, we have planned a set of methodology steps to be performed during the course of the graduate program. Those steps are represented in Figure 1. Each step may have one of three possible status: Done, In Progress or To Do.

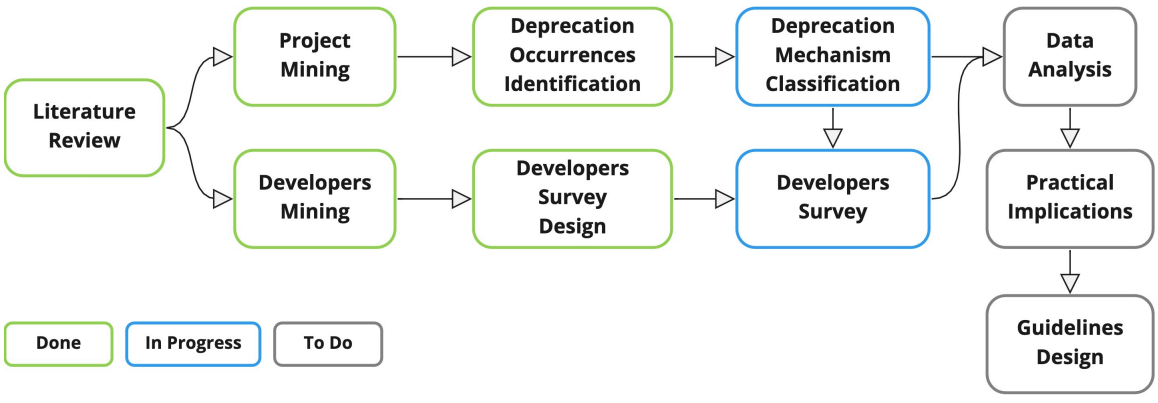

Figure 1. Diagram summarizing the work plan for this project.

As illustrated in Figure 1, the first step was a literature review, in which we focused on understanding the state-of-art of API deprecation and development practices regarding deprecation. We could observe several studies on API deprecation covering several popular programming languages [Robbes et al. 2012, Bogart et al. 2016, Brito et al. 2018, Li et al. 2018, Sawant et al. 2019]. However, we noticed that no previous work investigated the JavaScript ecosystem, which motivated us to study API deprecation on JavaScript projects.

The second and third steps were projects and developers mining. By analyzing JavaScript code bases, we were able to identify deprecation occurrences. To compose the required code base for this study, we selected the top-50 most popular JavaScript projects according to the number of `npm` downloads.[3]. *npm* is a well known package manager for JavaScript applications, which is a public collection of open-source JavaScript projects. Therefore, the *npm* website is an indicator of project popularity and their amount of client applications. *npm* states on their latest survey[4] that 99% of JavaScript developers rely on *npm* to ease the management of their project dependencies. This survey also points out the massive growth in *npm* usage that started about 5 years ago. After selecting 50 candidate projects, we downloaded their source code from GitHub, considering their latest stable version on November 20th, 2019.

To collect data on JavaScript API deprecation from developers perspectives, we searched for and randomly selected developers with contributions to JavaScript projects on GitHub. We filtered out developers with more than 100 followers, as very popular developers could be less likely to respond email surveys. We also removed developers with less than 50 contributions in the last year, as we could not ensure they had been actively working on JavaScript project recently. As a result, we currently found a sum of 14,480 developers.

To identify deprecation occurrences for further analysis, we performed a regular expression based search to find deprecation occurrence candidates. Every time one or more matches were found on a file, the file path and the code snippet were saved for further investigation. To support our analysis and the identification of API deprecation candidates in all 50 projects, we also used a JavaScript code parsing library, Flow[5], to find the context in which the API deprecation terms occur. We then exported the generated abstract syntax trees (ASTs) to manually analyze the deprecation occurrences. We sampled 20% of the abstract syntax trees and the respective code snippets for further manual investigation and classification of deprecation mechanisms.

Table 1 shows the five possible JavaScript deprecation cases we found in our analysis. We empirically derived these cases by manually and carefully analysing the samples of code snippets. If a certain occurrence does not fall in one of the proposed JavaScript deprecation mechanisms, we classify into the Others category.

Using the five deprecation mechanisms described on the classification step, we are currently designing a survey to collect data from developers regarding their opinions on JavaScript API deprecation and the mechanisms we previously identified. The current

[3]https://www.*npm*js.com/browse/depended

[4]https://cdn2.hubspot.net/hubfs/5326678/Resources/JavaScript%20Surveys/2019_*npm*_survey_FINAL.pdf

[5]https://flow.org/

Table 1. JavaScript deprecation mechanisms.

JS Deprecation Mechanism	Description
JSDoc	Use of the @deprecation JSDoc annotation
Code comment	Use of code comments excluding occurrences of JSDoc
Deprecation utility	Any sort of code function specially written to aid code deprecation at any complexity level
console.*	Use of the JavaScript engine native console API
Deprecation lists	List of deprecated elements
Others	Other adopted solutions

survey constitutes of six questions. The first three questions ask about developers experience on consuming deprecated APIs. Four and five, on the other hand, ask how often they deprecate APIs and which mechanisms they usually adopt. Lastly, the open-ended sixth question asks developers to share any thoughts, experiences or suggestions they might have regarding deprecation mechanisms on the JavaScript ecosystem.

Next step is data analysis, in which we we plan to combine all data collated from deprecation occurrences found by code source analysis and the survey results to answers this master research questions and summarize our understanding on JavaScript deprecation mechanisms and practices. By the end of this analysis, we expect to have some research outcomes such as understating of how common API deprecation is in JavaScript project and which deprecation mechanism or set of mechanisms are mainly used by developers. As secondary outcomes, we expect to describe characteristics of each mechanism, along with its advantages and disadvantages, and understating developers reactions to deprecation in JavaScript libraries. This analysis will provide us with inputs for the JavaScript deprecation guidelines design.

As a final contribution and practical implication of this master research, we expect to design and provide a set of JavaScript API deprecation guidelines, which will be heavily based on our empirical findings. We hope to describe the best practices, recommendations, techniques and important considerations when deprecating APIs in JavaScript.

4. Preliminary Results

We found deprecation occurrences in 29 (58%) out of the 50 analyzed projects on the deprecation occurrences identification step. The parsing tool extracted 1,279 deprecation contexts from the 214 files analyzed. From those, we selected a random sample of 268 cases (20%) for manual analysis. As presented in Figure 2, the most frequent deprecation mechanism is *deprecation utility*. This case represented 88 (32.8%) out of 268 cases. From those 88 occurrences, we detect that 75 contain replacement messages to support an API migration. Deprecation indicated by *code comments* represent 27 (10.1%) of the cases. This represents the usage of code comments excluding occurrences of JSDoc. Only 4 of those code comments contain replacement messages. The adoption of the *@deprecated JSDoc* annotation was identified 22 times (8.2%). However, only 10 of those occurrences have replacement messages. Deprecation elements described trough *lists* represent 6.7% of the analyzed sample (18 occurrences); 13 of those have replace-

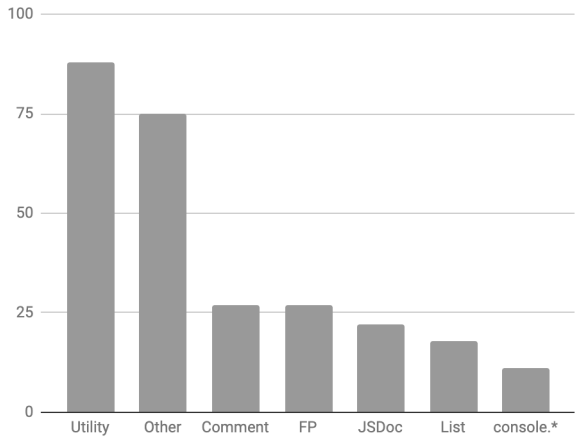

Figure 2. Deprecation mechanism occurrences per category.

ment messages. The direct usage of *console.** is the least present: 11 occurrences (4.1%), from which 10 have clear replacement messages to aid developers.

Overall, the analyzed sample suggests that deprecation adoption in not highly frequent in JavaScript APIs. However, 58% of all analyzed projects contain occurrences of deprecation in our study. Moreover, JavaScript projects deprecate their APIs using deprecation utilities, often throwing console warnings. This mechanism represented 32.8% of the studied sample. Using comments is also a common practice: considering both JSDoc and general code comments together, they represent 18.3%. From the categorized deprecation occurrences, we find that about 67% have replacement messages. However, those replacement messages are more common when the message is output to a console. To summarize, we can learn that there is no standard approach to deprecate JavaScript APIs, nor there is a single mechanism that is primarily used. Instead, we observe a few different approaches that are used alone or combined.

5. Conclusion and Next Steps

This master research proposal aims at investigating deprecation in the JavaScript ecosystem. This work can help developers better understand JavaScript API deprecation approaches and offer guidance on which mechanisms are more appropriate to a certain project context. Results of our first study [Nascimento et al. 2020], in which we investigated deprecation practices of 50 popular JavaScript projects, suggest that the use of deprecation mechanisms in JavaScript packages is low. However, we detect five different ways that developers use to deprecate APIs: deprecation utility, code comment, JSDoc, deprecation lists, and console messages. Among these solutions, deprecation utility and code comments are the most common practices. Finally, we find that the rate of helpful message is high. In this case, we detected that 67% of the deprecation occurrences have replacement messages to help an API migration.

We are following the work plan provided in Section 3 by deepening our analysis on deprecation mechanisms and conducting a survey with JavaScript developers. Finally, we expect to conclude the research with a cross-analysis of source code findings and survey results, and the proposal of a set of guidelines on JavaScript API deprecation best

practices to help and improve developers experience.

6. Acknowledgments

This research was partially supported by CNPq, CAPES, and FAPEMIG.

References

Bogart, C., Kästner, C., Herbsleb, J., and Thung, F. (2016). How to break an api: cost negotiation and community values in three software ecosystems. In *International Symposium on Foundations of Software Engineering (FSE)*, pages 109–120.

Brito, A., Valente, M. T., Xavier, L., and Hora, A. (2019). You broke my code: Understanding the motivations for breaking changes in apis. In *Empirical Software Engineering*, pages 1–35.

Brito, G., Hora, A., Valente, M. T., and Robbes, R. (2018). On the use of replacement messages in api deprecation: An empirical study. In *Journal of Systems and Software, vol. 137*, pages 306–231.

Granli, W., Burchell, J., Hammouda, I., and Knauss, E. (2017). The driving forces of api evolution. In *International Workshop on Principles of Software Evolution (IWPSE)*.

Li, L., Gao, J., Bissyandé, T. F., Ma, L., Xia, X., and Klein, J. (2018). Characterising deprecated android apis. In *International Conference on Mining Software Repositories (MSR)*, pages 254–264.

Moser, S. and Nierstrasz, O. (1996). The effect of object-oriented frameworks on developer productivity. In *Computer, vol. 29, no. 9*.

Nascimento, R., Figueiredo, E., Hora, A., and Brito, A. (2020). Javascript api deprecation in the wild: A first assessment. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 567–571.

Robbes, R., Lungu, M., and Röthlisberger, D. (2012). How do developers react to api deprecation?: the case of a smalltalk ecosystem. In *International Symposium on Foundations of Software Engineering (FSE)*.

Santos, A., Valente, M., and Figueiredo, E. (2015). Do javascript static analyzers detect bad coding practices? In *Workshop on Software Visualization, Evolution, and Maintenance (VEM)*.

Sawant, A., Robbes, R., and Bacchelli, A. (2018). On the reaction to deprecation of clients of 4 + 1 popular java apis and the jdk. In *Empirical Soft. Engineering, vol. 23*, pages 2158–2197.

Sawant, A., Robbes, R., and Bacchelli, A. (2019). To react, or not to react: Patterns of reaction to api deprecation. In *Empirical Soft. Engineering, vol. 24*, pages 3824–3870.

Tourwé, T. and Mens, T. (2003). Automated support for framework-based software. In *International Conference on Software Maintenance (ICSM)*, pages 148–157.

Xavier, L., Brito, A., Hora, A., and Valente, M. T. (2017). Historical and impact analysis of api breaking changes: A large scale study. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 138–147.