Mining Rule Violations in JavaScript Code Snippets

Uriel Campos, Guilherme Smethurst, João Pedro Moraes
Federal University of Pará
Belém, Brazil
{urielfcampos95, guitrompa1, joaopedromoraes94}@gmail.com

Rodrigo Bonifácio
University of Brasília
Brasília, Brazil
rbonifacio@unb.br

Gustavo Pinto
Federal University of Pará
Belém, Brazil
gpinto@ufpa.br

***Abstract*—Programming code snippets readily available on platforms such as StackOverflow are undoubtedly useful for software engineers. Unfortunately, these code snippets might contain issues such as deprecated, misused, or even buggy code. These issues could pass unattended, if developers do not have adequate knowledge, time, or tool support to catch them. In this work we expand the understanding of such issues (or the so called "violations") hidden in code snippets written in JavaScript, the programming language with the highest number of questions on StackOverflow. To characterize the violations, we extracted 336k code snippets from answers to JavaScript questions on StackOverflow and statically analyzed them using ESLinter, a JavaScript linter. We discovered that there is no single JavaScript code snippet without a rule violation. On average, our studied code snippets have 11 violations, but we found instances of more than 200 violations. In particular, rules related to stylistic issues are by far the most violated ones (82.9% of the violations pertain to this category). Possible errors, which developers might be more interested in, represent only 0.1% of the violations. Finally, we found a small fraction of code snippets flagged with possible errors being reused on actual GitHub software projects. Indeed, one single code snippet with possible errors was reused 1,261 times.**

***Index Terms*—Rule Violations; JavaScript code snippets; ESLinter**

I. Introduction

StackOverflow is the most well-known platform when it comes to asking and answering questions about computer programming [1]. Software engineers use StackOverflow on regular basis, and the range of questions asked is greatly diverse, varying from documentation [2], debugging [3], and even energy consumption [4]. In spite of this mixed landscape, answers on StackOverflow arrive in a blink of eyes; questions are answered in a median time of 11 minutes [5], [6]. Moreover, only 8% of the questions are left without an answer [5]. This reputation made StackOverflow an important place to go when developers are in stuck on a programming task, in doubt about the best solution, or even for learning purposes.

Indeed, software engineers take the answers suggested in the questions asked at StackOverflow seriously. As indicated in another study [7], developers tend to reuse verbatim copies or near verbatim copies of code snippets from StackOverflow. In particular, 20% of the code snippets from StackOverflow used on Github are employed without any modification. However, the ready use of code snippets from StackOverflow might hidden several concerns. For instance, the code snippets could be deprecated [8], misused [9], or even security flaws [10]. Software engineers under time pressure have little chance to thoroughly distill the code snippets they borrow from StackOverflow, which may have the potential of scattering such "low quality code snippets" over their software applications.

The ultimate goal of this work is to expand the understanding of the quality of StackOverflow code snippets. In particular, this work focuses on violations on JavaScript code snippets. We focus on JavaScript due to at least three important reasons: first, it is the *de facto* programming language for the web; second, it is the programming language with the highest number of questions tagged on StackOverflow; third, 65% JavaScript code snippets on StackOverflow are parsable and runnable [11], which makes them particularly attractive to be reused in practice.

To find JavaScript code snippets, we took advantage of SOTorrent [12], a dataset that curates the extraction and the evolution of StackOverflow code snippets. Through an automated process, we extracted 336k JavaScript code snippets for analysis. We relied on ESLint[1], an open source linter, to analyze the JavaScript source code and to flag programming errors, bugs, and stylistic coding standards. The main findings of this study include:

- We observed that violations are pervasive in our dataset of JavaScript code snippets. No single code snippet is free of violations. On average, our code snippets have 11.9 violations.
- We categorized the violations in terms of five rules employed by ESLint, namely: Possible Errors, Best Practices, Variables, Stylistic Issues, ECMAScript, and Node.js and CommonJs. Stylistic Issues account for 82.9% of the overall violations. Possible Errors, which developers might be more interested in, represent 0.1% of the violations.
- We found that only 36 code snippets out of the 6,303 ones with potential errors being reused on Github software projects. A total of 2,092 projects reuse these code snippets. However, only three code snippets are responsible for 94% of the overall reuse.

[1]https://eslint.org/

II. Background on ESLint

Different than full-fledged static analysis tools such as ESC/Java [13] or FindBugs [14], linters are a kind of lightweight static analysis tools that employ relatively simple analysis to flag non-complex programming errors, best practices, and stylistic coding standards. Several linters have been introduced for a variety of programming languages. This work is based on ESLint, the most downloaded and used JavaScript linter. By way of contrast to ESC/Java [13] and FindBugs [14], that leverage sophisticated techniques such as theorem proving and dataflow analysis to find bugs, respectively, ESLint relies on an Abstract Syntax Tree (AST) search. ESLint searches for code patterns (e.g., possible code errors or violations to coding standards) in the nodes of the AST tree. ESLint currently has a catalogue of 253 *rules*, which are specific code patterns (e.g., the use of a `return` statement on getters[2]) that are pattern matched in the AST. A *violation* happens when a rule is infringed.

III. Methodology

A. Research Questions

In this work we investigate the following important but overlooked research questions:

(**RQ1**) How commonplace are rule violations in JavaScript code snippets?

(**RQ2**) What are the most common rules violated in JavaScript code snippets?

(**RQ3**) Are JavaScript code snippets flagged with possible errors being reused in GitHub projects?

To answer **RQ1** we investigate the proportion of code snippets that contain any kind of rule violations. Our rationale is that accepted answers to JavaScript questions on StackOverflow might have been thoroughly designed and revised; then they might be less likely to have rule violations. For **RQ2** we distill the occurrences of the violations in terms of the six categories that we shall see in Section III-C. Finally, for **RQ3** we assessed whether violations related to potential errors, the ones that more interest software engineers [15], are reused in open source software projects on GitHub.

B. Curating the dataset

We started by selecting the questions tagged with JavaScript on StackOverflow. In our query, we placed two particular restrictions:

1) The questions should have an accepted answer, and
2) The accepted answer should have a code snippet.

When analyzing the code snippets, we perceived that some of them did not have a valid JavaScript source code (e.g., some of them have only HTML elements). In a manual investigation of a random set of code snippets, we noted that the ones that are not related to JavaScript were usually the ones with the fewer lines of code. We then decided to introduce an additional filter to remove code snippets smaller than 10 lines of code (LOC). This number was chosen after noticing that, on average, the selected code snippets have 7.2 lines of code (1st Quartile: 1 LOC, median: 4 LOC, 3rd Quartile: 8 LOC, max: 410 LOC).

Moreover, one interesting characteristics of the SOTorrent database is that it also keeps track of the evolution of the code snippets. This feature happens to introduce several code snippets duplication in our dataset (i.e., edited version of the code snippets. To avoid code snippets duplication, in this study we decided to focus only on the most recent version of the code snippets. After applying all these filters, we ended up with 432,146 JavaScript code snippets. When analyzing these code snippets, we also perceived that some answers have more than one code snippet. In that case, we merged the code snippets into a single file. After this procedure, we ended up with 336,643 code snippets, which comprehends our dataset.

C. Linting the dataset

We ran each one of the JavaScript code snippets using ESLint, an open source lint for JavaScript. ESLint has a set of rules that are used to assess the code quality attributes of a given source code. ESLint implements a total of 253 rules. Rules are grouped by categories, namely:

- **Possible Errors:** Rules relate to possible syntax or logic errors in JavaScript code. There are 36 rules available.
- **Best Practices:** Rules relate to better ways of doing things to help developers avoid problems. There are 72 rules available.
- **Variables:** Rules relate to variable declarations. There are 11 rules available.
- **Stylistic Issues:** Rules relate to style guidelines, and are therefore quite subjective. There are 91 rules available.
- **ECMAScript:** Rules relate to ES6, also known as ES2015. There are 31 rules available.
- **Node.js and CommonJS:** Rules related to code running Node.js, or in browsers with CommonJS. There are 11 rules available.

Besides these rules, there are still one Strict Mode rule (related to strict mode directives), 11 Deprecated rules (rules have been deprecated in accordance with the deprecation policy, and replaced by newer rules), and 18 Removed rules (rules from older versions of ESLint (before the deprecation policy existed) have been replaced by newer rules).

In our study, we employed the *Standard* ESLint configuration. We chose this configuration because it is based on guidelines curated at https://standardjs.com and implemented by several linters, including ESLint. This configuration uses 117 rules (Possible Errors: 25, Best Practices: 36, Variables: 6, Stylistic Issues: 34, ECMAScript6 : 13, Node.js and CommonJs: 3). After running the ESLint locally over the 336k JavaScript code snippets, we created python scripts to read and perform analysis over ESLint output. All data created in this work is available online for replication purposes[3].

[2]https://eslint.org/docs/rules/getter-return

[3]Available at https://github.com/urielfcampos/linting-js-codesnippets

IV. Results

In this section we present our results grouped by the research questions.

RQ1. How commonplace are rule violations in JavaScript code snippets?

When analyzing the presence of violations in our dataset, our first observation was that just one of the 336,643 studied code snippets was free of violations (stackoverflow.com/q/51594048). However, a closer look at this particular code snippet revealed that the author who created it intentionally suppressed all warnings raised by ESLint by commenting the code with the `/*eslint-disable*/` instruction. We manually removed this comment and reran ESLint on this code snippet. This new execution reported 96 violations. Therefore, the first finding of this work is that no single code snippets in our dataset was free of violations. On average, the studied code snippets have 11.94 violations. Figure 1 shows the distribution of violations in our dataset.

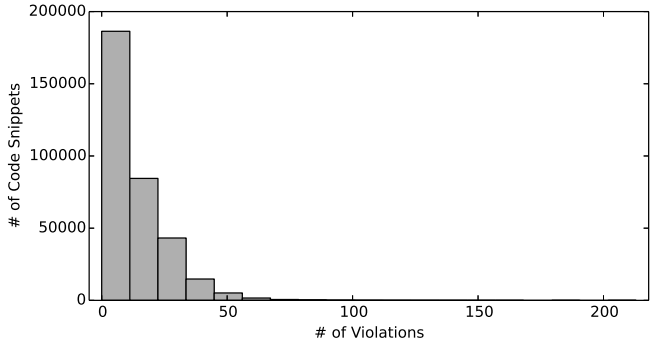

Fig. 1. The violations distribution in the code snippets dataset.

As we can see in this figure, the majority of the code snippets have between one and 50 violations (1st Quartile: 1, median: 9, 3rd Quartile: 20). At one extreme example, one code snippet violated 53 rules, totalizing 213 violations (the same rule could be violated more than once). Overall we found 5,587,357 violations. When talking about the number of rules violated, on average, each code snippet violated five rules (1st Quartile: 1, median: 4, 3rd Quartile: 8, max: 53).

RQ2. What are the most common rules violated in JavaScript code snippets?

We now break down the 5,587,357 violation found in terms of their categories. Figure 2 shows the results.

As we can see, the **Stylistic Issues** category dominates with 4,632,348 violations (82.9%). It is important to note that, since we have to merge different code snippets into a single one, we removed 3,461,739 violations in indentation rules (which fall in the **Stylistic Issues** category. Indentation violations were by far the most common one in our dataset. Following next is the **Variable** category, with 787,824 violations (14.1%), then **Best Practices (BP)** with 157,578 violations (2.8%), **Possible Errors (PE)** with 6,303 violations (0.1%), **Node.js/Common.js (N/C.js)** with 3,304 violations (0.05%), and **ECMAscript 6 (EC6)** with 548 violations (0.009%). We now discuss the three most common violations per category.

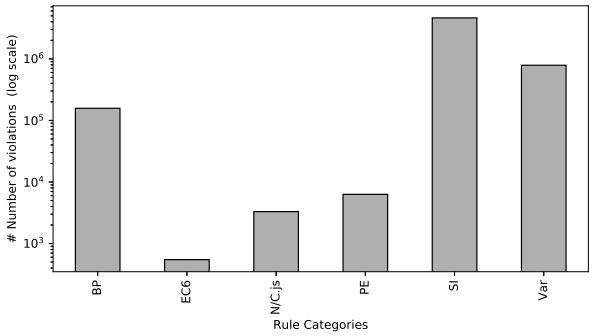

Fig. 2. Number of violations per rule category. BP stands for Best Practices, EC6 stands for ECMAscript 6, N/C.js stands for Node.js/Common.js, PE stands for Possible Errors, SI stands for Stylistic Issues, and Var stands for Variable.

Stylistic issues: The three most common violations under this category are: *semi* (1,477,808 occurrences), *quotes* (700,770 occurrences), and *no-trailing-spaces* (374,012) occurrences). The *semi* rule, which requires (or disallows) the use of a semicolon at the end of each statement, accounts for 32% of the overall stylistic issues.

Variables: The three most common violations under this category are related to (1) the use of undeclared variables (*no-undef*, with 719,679 occurrences), (2) the presence of unsused variables (*no-unused-vars*, with 67,816 occurrences), and (3) the initialization of variables to `undefined` (*no-undef-init*, with 150 occurrences). As we could see, *no-undef* concentrate the variable violations (91% of them).

Best practices: The three most common rules violated under this category are: the *no-multi-spaces* rule, that is, the use of multiple spaces in a row, but not for indentation purposes (54,768 occurrences), the *eqeqeq* rule, that is, the use of regular equality operators == and !=, instead of their type-safe counterparts === and !== (53,321 occurrences), and the *curly* rule, that is, the omission of curly braces when a block contains only one statement (14,989 occurrences).

Possible Errors: The three most common violations under this category are: *no-irregular-whitespace* (2,037 occurrences), *no-cond-assign* (910 occurrences), and *no-unreachable* (485 occurrences). Interestingly, the most common possible error is using invalid or irregular whitespace (e.g., mixing tabs and spaces), which have a fairly easy fix: rewrite the line statement from scratch.

Node.js/Common.js: The three most common violations under this category are: *handle-callback-err* (2,855 occurrences), *no-path-concat* (444 occurrences), and *no-new-require* (5 occurrences).

ECMAscript 6: The three most common violations under this category are: *template-curly-spacing* (164 occurrences), *no-useless-constructor* (154 occurrences), and *no-this-before-super* (88 occurrences).

RQ3. Are JavaScript code snippets flagged with possible errors being reused in GitHub projects?

In this research questions, we selected only the 6,303 code snippets that violate at least one of the 25 **possible errors** rules. We decided to focus on these kind of code snippets because they might be the ones that most interest developers, since they could lead to software errors [15].

We used SOTORRENT to match whether these code snippets are being used on GITHUB projects. We found 36 JavaScript code snippets flagged with possible errors (0.11%) being reused on 2,092 GITHUB projects. One important observation, however, that due to the forkability characteristics of GITHUB projects, most of these 2,092 GITHUB projects are actually fork projects (which might enjoy the same architecture and file content of the forked project). In fact, only 845 out of the 2,092 GITHUB projects (40%) are non-forks. A similar finding was reported in another context [10]: a small number of code snippets are proliferated into thousands of software projects.

When manually analyzing the frequency of use of these 36 code snippets on the GITHUB projects, we perceived that only three of them are responsible for 94% of the reuse on GITHUB projects. For instance, the code snippet stackoverflow.com/a/9039885, which violated seven rules totalizing 22 violations, was used 1,261 times. In this particular code snippet, the intention was to detect whether the browser is using an iOS operating system. Due to the widespread presence of iOS devices, this feature that might be needed in many web applications, which could partially justify this high number of occurrences. Moreover, code snippet stackoverflow.com/a/7513356, which violated 15 rules, totalizing 64 violations, was used 397 times. This code snippet is intended to control the iframe of Youtube players, also a feature that might be very common in the web development arena. Finally, the code snippet stackoverflow.com/a/5598797, which violated 8 rules, totalizing 19 violations, was used 325 times. This code snippet was intended to determine the position of HTML elements relative to the browser window. Similarly, we also believe that this feature might be of interest of many web applications. One final observation about these three reused code snippets: *semi* was the most common rule violate in all of them.

V. IMPLICATIONS & LIMITATIONS

Implications. Based on our results, we believe that platforms such as STACKOVERFLOW could integrate static analysis tools to assess quality attributes of the source code posted on there. This in turn could feed the gamification mechanism of these websites, making not only readers more aware of any potential concerns that may exist in the code snippets, but also fostering a STACKOVERFLOW users to create questions/answers with better quality. Moreover, by showing the rule violations is pervasive in JavaScript code snippets, CS professors should assert to students that they cannot ignore it. Still, professors can take advantage of our curated list of JavaScript code snippets and discuss their violations in the classroom. Since ESLINT also provides high-level suggestions on how to fix the violations, professors could use them to discuss students' suggestions on how would they fix the violation while presenting the suggestion provided by ESLINT.

Limitations. First, we only employed one linter, ESLINT. Although ESLINT is one of the most popular ones [15], it clearly did not catch all kinds of code violations (although it is the one with the highest number of rules supported). Moreover, in this study we employed 117 rules that cover a wide spectrum of source code violations. Second, we restricted our analysis to code snippets longer than 10 lines of code. This decision was taken to avoid non-JavaScript code. In a manually analysis of a random sample of 100 code snippets we found that only five of them (31041975, 33367225, 24694986, 15714480, 31363059) did not have any JavaScript code. Furthermore, we did not consider violations related to indentation (we removed 3,461,739 violations related to indentation). This decision was taken because we merged answers that have more than one JavaScript code snippet into a single file. When doing so, we may have introduced indentation issues. However, due to the scale of our analysis (hundred of thousand of code snippets), manual analysis to fix the indentation errors was unfeasible. Finally, linters are mainly adopted for analysing the full program. In contrast, most code snippets in Stack Overflow are partial programs. This may justify why many code snippets have variable violations.

VI. RELATED WORK

There is a recent flow of studies trying to shed some light on quality concerns hidden in STACKOVERFLOW code snippets [8], [9], [10]. Some other studies focused on understand how developers perceive static analyzers, in general [16], and linters, in particular [17], [15], [18]. More generally, there are several works on STACKOVERFLOW, covering different aspects of software maintenance and evolution [2], [3], [4], [5], [6]. However, to the best of our knowledge, none of them deal with the respect of understanding how common are rule violations JavaScript code snippets or whether code snippets flagged with potential errors are used in GitHub projects.

VII. CONCLUSIONS

In this work we mined rule violations in 336,643 JavaScript code snippets. Our analysis was based on ESLINT, a popular open source JavaScript linter. When employing this linter in our corpus of code snippets, we perceived that no single code snippet is free of violation (on average a code snippet have 11.9 violations). In terms of the caracteristics of these violations, we found that the majority of them are related to stylistic issues, such as the absence of a semicolon at the end of the statement or the lack of a consistent use of quotes. Rule violations that might incur in errors are responsible for only 0.1% of the overall violations. Still, we perceived that only 36 code snippets with possible errors were used on GITHUB projects. However, we found one single code snippet being used 1,261 times.

ACKNOWLEDGMENTS

We thank the reviewers for their helpful comments. This work is partially funded by PROPESP/UFPA and CNPq (#406308/2016-0).

REFERENCES

[1] Jay Hanlon. Five years ago, stack overflow launched. then, a miracle occurred. https://stackoverflow.blog/2013/09/16/five-years-ago-stack-overflow-launched-then-a-miracle-occurred/, visited 2019-02-06.

[2] Chris Parnin and Christoph Treude. Measuring api documentation on the web. In *Proceedings of the 2Nd International Workshop on Web 2.0 for Software Engineering*, Web2SE '11, pages 25–30, 2011.

[3] Fuxiang Chen and Sunghun Kim. Crowd debugging. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 320–332, 2015.

[4] Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 22–31, 2014.

[5] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2857–2866, 2011.

[6] A. Bosu, C. S. Corley, D. Heaton, D. Chatterji, J. C. Carver, and N. A. Kraft. Building reputation in stackoverflow: An empirical investigation. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 89–92, May 2013.

[7] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? *Empirical Software Engineering*, Jul 2018.

[8] Jing Zhou and Robert J. Walker. Api deprecation: A retrospective analysis and detection method for code examples on the web. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium*

[17] Kristín Fjóla Tómasdóttir, Mauricio Finavaro Aniche, and Arie van Deursen. Why and how javascript developers use linters. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, pages 578–589, 2017.

on Foundations of Software Engineering, FSE 2016, pages 266–277, 2016.

[9] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. Are code examples on an online q&a forum reliable?: a study of API misuse on stack overflow. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 886–896, 2018.

[10] F. Fischer, K. Bttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl. Stack overflow considered harmful? the impact of copy amp;paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 121–136, May 2017.

[11] Di Yang, Aftab Hussain, and Cristina Videira Lopes. From query to usable code: An analysis of stack overflow code snippets. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 391–402, 2016.

[12] Sebastian Baltes, Christoph Treude, and Stephan Diehl. SOTorrent: Studying the origin, evolution, and usage of stack overflow code snippets. In *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, Montreal, Canada*, 2019.

[13] Cormac Flanagan, K. Rustan M. Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, and Raymie Stata. Extended static checking for java. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, PLDI '02, pages 234–245, 2002.

[14] David Hovemeyer and William Pugh. Finding bugs is easy. *SIGPLAN Not.*, 39(12):92–106, December 2004.

[15] K. F. Tmasdttir, M. Aniche, and A. Van Deursen. The adoption of javascript linters in practice: A case study on eslint. *IEEE Transactions on Software Engineering*, pages 1–1, 2018.

[16] Maria Christakis and Christian Bird. What developers want and need from program analysis: an empirical study. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, pages 332–343, 2016.

[18] Moritz Beller, Radjino Bholanath, Shane McIntosh, and Andy Zaidman. Analyzing the state of static analysis: A large-scale evaluation in open source software. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*, pages 470–481, 2016.