JavaScript Coding Guideline Compliance on Stack Overflow

Chizzy Meka
[Department]
[University Name],
[University Address & City],
[Postcode].
[Email Address]

ABSTRACT

Context: Several research studies have criticised the code quality on Stack Overflow. There have been inferences that their effects subtly manifest in the form of unreliable and insecure software systems and poor programming practices amongst developers.

Objective: This study aims to evaluate the extent of compliance of JavaScript snippets on Stack Overflow.

Method: Using the SOTorrent Dataset, we pulled JavaScript snippets on Stack Overflow and reconstructed them into JavaScript source code files to analyse various insights.

Results: We found that the snippets have an error violation ratio of 0.95 and a 0.75 security violation ratio. We also found a weak correlation between compliance and post scores and a weak correlation between compliance and user reputation.

Conclusion: We do not recommend Stack Overflow for JavaScript developers looking to improve their conformance to guidelines. We also urge users looking for code examples to strive to understand the code snippets before proceeding any further.

1 Introduction

1.1 Programming Style

A programmer's approach to source code writing (programming style) hugely impacts how easy it is to read and maintain their code. Formally put, Programming Style is a collection of rules or guidelines applied when writing computer source code [4].

1.1.1 Why Establish a Programming Style?. Programming Style aims to help make computer software source code more readable and easier to maintain. It tries to achieve these objectives from five points, including [1][7]:

- Readability: Computer source code should be made as easy as possible to read.
- Clarity of Intent: The author's/programmer's intention of a piece of code should be easily decipherable from the code.
- Clarity of Structure: It should be easy to comprehend how a software program fits in together.
- Correctness and Maintainability: Programming Style should discourage the duplication of code in a program.
- Modularity: Programming Style should promote reusability by ensuring that different classes or functions manage different concerns.

1.1.2 Programming Style Compliance. The section above infers that the concept of Programming Style Compliance involves ensuring that a piece of source code conforms to a particular trusted set of rules. The programming language creator in question or another entity of authority related to the subject programming language could publish these guidelines. As a recommendation, individual programmers should form their programming style (under the umbrella of some programming style guidelines of a given language) and stick with it to maintain consistency [3].

These programming guidelines govern different aspects of a programming language. These can include general code layout, strings, whitespaces in expressions and statements, trailing commas, comments, naming conventions and other miscellaneous practical programming recommendations [4].

1.2 Community Question Answering

The community question answering concept has been popular for several years since the maturity of the internet. These web platforms allow community members to post questions that traditional information retrieval systems such as Google Search or Bing cannot readily address. Some of the most popular community question answering platforms are Yahoo! Answers and Quora. In fact, in recent years, community question answering platforms have become so popular that there are now dedicated community question answering platforms for specific topics [19]. The Stack Exchange Network is one example of such community question answering platforms that offer many sub platforms covering a particular topic. These topics include programming, software engineering, computer usage, Linux OS, information security and even non-computing related issues such as the English language, bicycles, vehicle maintenance, movies, and TV. Under this network, the sub-website dedicated to programming is called Stack Overflow [23].

1.2.1 Stack Overflow. Stack Overflow is a question and answers online platform intended for programmers – both professional and hobbyists. It covers a variety of programming-related topics [8].

Stack Overflow encourages active involvement by enabling users to vote posted questions and answers. Users can vote 'up' (for

positive) and ‘down’ (for negative). Users can also modify these questions and answers, if necessary [8].

The website rewards users for their contributions through reputation points. It allows users to attain additional voting ability, commenting ability and ability to modify other users’ contributions. This reputation points system somewhat gamifies the platform [8].

This report aims to assess the degree of compliance with the JavaScript language’s standard programming style amongst Stack Overflow users. The paper sets out to determine the extent to which JavaScript code snippets on the platform comply with recommended programming styles. It will also examine the most frequently violated programming style rules; ascertain whether favourably received posts tend to comply with programming guidelines; and whether reputed users on the platform tend to abide by these guidelines.

The research experiment will replicate the methodology of a similar study on Python by Bafatakis, N. *et al.* [13] and closely benchmark the two languages’ results.

Our study features core research questions that will help us achieve the feats stated above.

1.3 Research Questions

1. Do JavaScript code snippets on Stack Overflow usually conform to programming style guides?
2. Which is the most violated JavaScript programming guideline on Stack Overflow?
3. Are compliant JavaScript posts consistently positively voted?
4. Do highly reputed users follow programming style guides more than lowly reputed users?
5. Do JavaScript code snippets on Stack Overflow usually conform to secure programming standards?

2 Background

2.1 Motivation for Study

There has been significant research in the past that concluded that code snippets on Stack Overflow suffer from many flaws which adversely affect their quality [13] [16] [17]. Many papers have discussed the decline in code excellence and persistence of poor-quality code on the Stack Overflow platform. One of the most common reasons is the contribution of code by inexperienced contributors, of which most times, these contributors do not even understand the code [17].

There is also the finding that constant and rapid evolution of technology perpetually and continually leads to outdated code. Users commonly rely on commenting or making additional posts to raise concerns about any provided solution, which may no longer be working. This practice is because Stack Overflow does not offer version control or bug tracking mechanism for managing code snippets. This circumstance affects the reliability of the code obtained from the platform [18].

Another finding inferred that user interaction drives the entire Stack Overflow process. Its community’s behaviour – dubbed the human factor - heavily influences the platform. The research found that there are broadly three ‘undesired’ groups of users primarily responsible for the web platform’s poor-quality code. These include users who habitually ask many low-quality questions and prompt trivial answers that overload the posts with low-quality content. This phenomenon makes it more difficult to find excellent solutions. In the second category are the users who continually pose questions without exerting any prior effort to find their answers through search engines or consulting technical documentation. The study describes this user category as ‘Help Vampires'—a term based on the notion that their questions are often tedious and usually duplicates. Also, these users are only interested in obtaining solutions to their problems without positively contributing to the community in return. The third category comprises the ‘reputation boosters’ or ‘feeders’ whose sole purpose is to answer any question as quickly as possible to secure the reputation points attached to the problem. While these users may occasionally offer high-quality solutions, their responses usually range from low-quality to mediocre. Their primary incentive to interact on the platform is to collect reputation points, not provide high-quality answers. This behaviour plays directly into the Help Vampires’ pursuit. Together, these two latter categories keep multiplying the low quality of code snippets on Stack Overflow at a very exponential rate [19][20].

The most notable technical and (sometimes ethical) effects of these factors manifest in several ways. Some of these include software security vulnerabilities. The Stack Overflow platform faces accusations of encouraging the development of insecure software systems. This accusation state that the platform allows inexperienced programmers to copy code snippets that they barely understand and integrate them into their projects. This practise continues even though these programmers may not fully comprehend these snippets’ ramifications on their applications’ functionalities [17].

Another notable technical effect is the loss of cohesion in recipient software systems. This effect is a scenario whereby an application or the specific class that receives code snippets from Stack Overflow tends to fare poorly. This phenomenon is observable when the system’s cohesion is measured using the Low-level Similarity-based Class Cohesion (LSCC) and Class Cohesion (CC) metrics. Research also discovered that this cohesion problem persists in the system over the recipient software’s evolution [16].

Finally, there is also the ethical issue of licensing whereby programmers (in most cases) inadvertently fall afoul of license conditions. This breach tends to happen when they copy and paste questions and answers without paying attention to the terms governing the use of code obtained from Stack Overflow [21].

These findings on the questionability of the code quality on Stack Overflow birthed the need to investigate this apparent low-quality content further to determine the extent to which it conforms with (or violates) standard programming styles. The study chose to

approach this investigation from the programming style standpoint because of the cruciality of programming style compliance to practical programming and the allied benefits, as addressed earlier. Thus, the degree of compliance with programming style (or lack thereof) will provide invaluable insight into the extent of the seemingly low-quality code epidemic plaguing this popular Q&A platform.

The work of Bafatakis, N. *et al*. 2019, solely studied the code compliance level of the Python scripting language on Stack Overflow. This study extends their effort by investigating the programming style compliance of an alternate well-known scripting language – JavaScript - and comparing both findings.

2.2 The JavaScript Language

The JavaScript language is an ECMAScript specification [5] conforming scripting language created in 1995. Some of its characteristics include a high-level, just-in-time compiled and multiparadigm design. It started as a scripting language for web pages. However, its usage has led to the creation of many frameworks, which has seen the JavaScript language spread beyond the browser environment, for example, Node.js, Apache CouchDB and Adobe Acrobat [6].

2.3 Python

The Python language is a general-purpose, high level, interpreted programming language first released back in 1991. The language has risen to prominence for creating backend logic on servers for web applications. Generally considered simple and easy to read, the Python language has gained fame as a beginner programmer's introductory programming language. As a result, it is now a popular choice in universities and educational institutions for teaching students the basic concepts of programming [22].

2.4 Why JavaScript?

According to TIOBE Index, a website that rates the popularity of programming languages, JavaScript has remained one of the top programming languages since the early 2000s, hovering between the 12th position in October 2014 February 6 2019 [10]. It sits at the 7th position as of late 2020 [15].

On the other hand, the TIOBE Index stats convey that since the early 2000s, Python hit its highest position (3rd) in late 2020 and its lowest position (13th) back in 2003 [14]. It currently sits at the 3rd position as of late 2020 [15].

Alongside PHP, which is currently at the 8th position [15], JavaScript and Python are the only other scripting languages that occupy the top ten places in the top programming languages TIOBE Index. This attribute makes these languages ideal candidates for this type of study.

JavaScript also remains a popular language on Stack Overflow. Figure 1: Growth of JavaScript vs Python [9] shows its growth relative to Python on the Q&A platform.

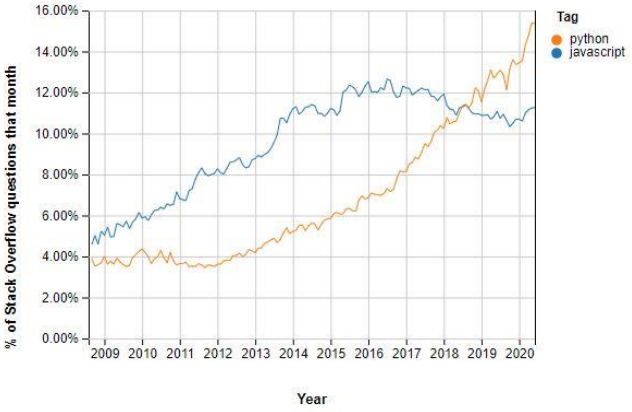

Figure 1: Growth of JavaScript vs Python [9]

According to custom queries ran on Stack Exchange Data Explorer [12], there are currently around 19,810,294 tags and a total of about 20 million questions on Stack Overflow [11]. There are now approximately 2,047,385 questions linked to the JavaScript tag as of late 2020. In other words, one tag, that is, *'javascript'*, out of a total of 19,810,294 tags on Stack Overflow, is related to around 10% of the total number of questions on the platform as of late 2020.

Similarly, the *'python'* tag relates to around 1,493,738 questions, which means that about 7.5% of Stack Overflow questions are Python-related.

Another reason for nominating is that JavaScript linters, like Python, can assess partial and incomplete code lines, making the language suitable for our purpose. This attribute is beneficial because it is a typical practice on community question answering platforms for question answerers to provide a few code lines in response to programming enquiries.

3 Literature Review

This section reviews relevant literature about the Stack Overflow platform's vital role in modern software development; the code quality issues rampant on the forum, including some of its causes and effects. It also covers programming style and rule violations of coding standards and conventions.

Regarding the Stack Overflow platform, it is imperative to review some relevant literature to help understand the central role that the platform plays in the programming community and the potential (effects) aftermath it can exert on software projects. Studies have shown that most modern software projects contain code examples copied from Stack Overflow. Developers, novices and experts alike modify these code examples and fit them into various recipient applications [29]. This finding highlights the indispensability of Stack Overflow to the modern programmer.

In a study to determine the practical use of Stack Overflow, in 2017, Abdalkareem, R. *et al*. analysed 1414 source code commits related to Stack Overflow. They discovered that modern programmers use the site extensively to support software

development tasks and garner feedback on their solutions. They also found that the platform was beneficial in software development tools usage, programming languages and, to a lesser degree, web frameworks [28]. However, they noted that this resourcefulness also comes with non-trivial costs.

Lopez, T. *et al*. carried out a qualitative research study in 2008 to ascertain the extent to which programmers rely on Stack Overflow for solving software security problems. They employed a set of software security-related questions and evaluated them for patterns. They found out that software security discussions were vastly abundant on Stack Overflow. Additionally, the study's findings also indicated that the platform greatly influences the software security views and opinions amongst non-specialist programmers [24].

In 2016 Acar, Y. *et al*. carried out a study to investigate how secure code snippets from the Stack Overflow platform are. Their experiment surveyed 295 android developers who had published Android apps on the Google Play Store. They enquired about their use of resources in tackling and solving software security issues. They designed and executed a second experiment with the obtained results. They had participants write security and privacy-intensive apps within a time-constrained period. The participants could only consult one type of resource, including Stack Overflow, the official Android documentation, or books on software development. Their study found that the participants that worked with Stack Overflow developed profoundly less secure solutions. Further analysis of the 139 Stack Overflow resources that the participants used in the experiment revealed that about 25% of them were only helpful in solving the tasks. A meagre 17% of the code snippets demonstrated acceptable secure software development practices [17].

In another software security related study, Yang, X. *et al*. researched in 2016 to determine the most popular types of software security-related questions that programmers ask on Stack Overflow. Their research employed natural language processing techniques to analyse a Stack Overflow related dataset. They discovered that the most common software security-related questions are password hashing and SQL injections [38]. Thus, it is alarming that despite using Stack Overflow to discuss and exchange ideas on such critical topics, code snippets' security on Stack Overflow remains questionable.

At this juncture, we have highlighted Stack Overflow's indubitably dominant position as a resource centre for programmers. We have also assessed the resultant potential effects its resources can have on software projects. It is logical to state that if developers are willing to make up their perceptions and convictions on critical topics like software security and code quality based on the community interactions they engaged in on Stack Overflow, then any decent effort to ascertain the quality of code obtainable there cannot be understated.

In 2019, Ahmad, M. and Cinneide, M. set out to understand the effects of copying and reusing code from Stack Overflow on a software system over time. To this end, using the SOTorrent Dataset, they employed GitHub projects which featured Java code snippets obtained from Stack Overflow. They iteratively measured the quality (using Low-level Similarity-based Class Cohesion (LSCC) and Class Cohesion (CC) metrics) of the receiving class. These measurements took place before and after integrating the Stack Overflow code snippet and at a later date during the development of the project. They found out that the copied code snippet adversely affected about 70% of their sample, made up of 378 classes. The study also found out that this adverse effect lingered over those software systems' future evolution [16].

The succeeding literature considers the quality issues, and the general quality observed in the code snippets obtainable from Stack Overflow. The reviewed literature highlights the importance of maintaining a high level of vigilance.

Ginsca, A. and Popescu, A. carried out an investigative study in 2013 to determine ways to automate the retrieval of high-quality answers from collaborative web applications such as Stack Overflow. Using a large Stack Overflow dataset, they set out to assess user profiles of contributors. They tried to discern any aspects of these profiles or their activity that indicate that the user is an active contributor. Their experiment found a significant correlation that suggested a directly proportional relationship between the completion of user-profiles and high-quality contributions. The users with a more completed profile tended to contribute better high-quality content [30].

In 2015, Jin, Y. *et al*. studied the effects that the gamification of Stack Overflow could be having on code quality. Their study assessed the distribution of gamification tendencies on the platform by analysing 101,291 posts over four months. They reported an average response time of 327 seconds. They concluded that users might, at times, commit to responding quickly instead of ensuring they provide quality content [31].

Anand, D. and Ravichandran, S. approached the speedy response vs code quality concerns differently. In their 2015 research study, they reported that a Q&A platform like Stack Overflow that operates by incentivising users to contribute content via the implementation of a voting system might be deeply flawed. They deduced that, firstly, Stack Overflow might be successful in encouraging participants to contribute quality content. Secondly, many a time, the voting system tends to favour questions based on popular topics such as popular programming languages like Java or HTML. As a result, many users may vote up trivial questions because they can quickly consult the relevant documentation to confirm the solution and subsequently give a positive vote to the post. In contrast, high-quality problems tend to warrant high-quality answers. This phenomenon leads to fewer votes and attention because the participants do not want to put in the required effort [32].

The review thus far largely agrees with the findings by Srba, I. and Bielikova, M. [19] addressed under the 'Background' section. In which they investigated the mitigating factors that are causing Q&A sites such as Stack Overflow to fail. Using Stack Overflow as a case study, Srba, I. and Bielikova, M. assessed community perceptions of the user categories on the site and, in harmony with Jin, Y. *et al*. study earlier, they found a unique subcategory of an 'undesired' group. Their sole motivation is to provide very speedy answers without paying attention to quality.

Asaduzzaman, M. *et al.* employed a Stack Overflow dataset to categorise unanswered questions on the platform to determine the reasons why these questions remained unanswered. They found that toxic users constitute about 11.75% of the fully or partially unanswered questions after conducting their analysis. These users do not care about quality, and their actions result in overall low-quality content on the site [33]. This 2013 study further attests to the existence of an 'undesired' group on Stack Overflow.

Baltes, S. *et al.* in 2018, when they analysed how the content on Stack Overflow evolves by creating the SOTorrent Dataset. This Dataset derived from the formal SO data dump grants accesses to version history information at the post level, individual text level or code block level. This study highlighted the need for users to keep posts updated to ensure that code snippets will evolve in tandem with newer software and library releases. The study also highlighted the incompleteness of human action amongst Stack Overflow users. [18].

Similarly, Ragkhitwetsagul, C. *et al.* carried out a clone detection study on Stack Overflow to determine their degree of malignancy. They executed a survey of 201 users with high reputation points. Their research found that 65% of these users (131) have never been called upon to update any outdated code snippet. Around 20% of these users (26) have never (or only partially) fixed any code snippets [34].

As can be observed from the review of various previous work, there have been many approaches to assessing and evaluating code quality on the Stack Overflow. These include (but not limited to) the human factor, software security, software metrics and lastly, programming style - which we will cover in this study.

On the topic of programming style, programmers, and software professionals, in general, have always recognised the need to develop human-readable code. Anecdotally speaking, there is significant decades-old literature that evidences this statement - as per the next few reviews.

In 1974, Kernighan, B. and Plauger, P. of Bells Laboratories surveyed specific programming style aspects, including basic expressions and structure. Using verbatim examples from relevant textbooks of the period, they showcased the effects of coding style violations and recommended improvement strategies [27]. The same study concluded that programs can be functionally similar yet stylistically dissimilar.

Many coding standards have been proposed for different programming languages over the past few decades to promote readable computer programs' authorship. VanNeste, K. designed and developed coding standards and conventions for the Ada programming language in 1986. This document covered different aspects of the language, including identifiers, indentation, alignment, spacing, comments, control structure implementation and programs units [26].

Similarly, in 1993 Shannon B. of Sun Microsystems developed a set of coding standards and recommendations for the C programming language. The development team of the source code for Oracle's Solaris operating system applied these guidelines. It assisted the programmers in sharing their source code and facilitated the development of programming tools. The standards covered file naming conventions, program organisation, file organisation, header files, indentation, comments, declarations, statements, white spaces, use of parenthesis, naming conventions, continuation lines, constants, the 'Goto' statement, variable initialisation, multiple assignments, pre-processor, portability, working with both the Lint and Make programs as well as project-based/individual coding standards [35][36].

Indian Hill lab by AT&T introduced the Indian C Style in 1990 and amended it in 1997. The programming style addressed three broad areas. Firstly, the proper use of comments and white space for better code structure and the appropriate use of expressions, statements, and functions to aid comprehensiveness. Secondly, designing code such that it is easily maintainable and portable to obscure computer systems. The third area addressed sticking to a coding style (on an individual level) consistently and avoiding mixing programming style, a practice that is deemed worse than any single 'bad' programming style [3]. The 'Programming Style Compliance' section under the 'Introduction' section of this report also addressed this third aspect. However, while this notion is strongly encouraged amongst software professionals, the reality is vastly different, as supported by relevant literature.

Wiese E. *et al.* in 2017 conducted a research study on how to teach new programmers to recognise and implement correct coding style and concluded that this objective was difficult to achieve. They reported that even though new programmers improved their recognition of decent coding style, they still found it challenging to implement style improvements during practice. This observation held accurate even with a purpose-built computer-based coding style tutor, AutoStyle, which assisted learners by recommending complex functions [25].

Li, X. and Prasad, C. implicitly supported the findings above in their 2005 study. The research investigated students' outlook on learning and adopting conventional programming styles. They carried out an extensive questionnaire-based survey study and recorded observations made during lecture sessions. They found that many novice developers are aware of the benefits of conforming to common sense Programming Style. However, they usually tend to not adhere to it in practice [2].

As stated earlier, this study is concerned with the compliance of these coding styles. It takes inspiration from two notable studies that address programming style violations on Stack Overflow.

The first research was the 2019 study by Bafatakis, N. *et al.*, which employed the evaluation of coding style in assessing the quality of code snippets on Stack Overflow. In their research, they examined the compliance level of Python code snippets on Stack Overflow. Their experiment analysed 1,962,535 code snippets tagged with the *'python'* tag and selected snippets with a minimum of six statements of the Python language. The study showed that 93.87% of the nominated snippets violated the Python coding style with a median value of 0.7 violations-per-statement. They also reported no clear correlation between a user's reputation points and correct programming style. However, they observed a relationship between the vote score for a given post and the average number of violations in the post [13]. This research study is a direct product of their 'Future Work'

suggestion, which proposes similar experiments on other programming languages.

Another inspirational study that motivated this research was by Ferreira Campos, U. *et al*. In 2019, they set out to assess the rule violations of JavaScript code snippets on Stack Overflow. To achieve this objective, they mined about 336,643 JavaScript code snippets from the answers provided for JavaScript-related questions. They proceeded to statistically analyse these snippets using the popular JavaScript static analysis tool, ESLint. After the experiment, they found out that all the JavaScript code snippets on the entire Stack Overflow platform had violated coding standards somehow. They reported an average of 11 violations per snippet. The study findings also indicated that most contravened rules were stylistic in nature. These included missing semicolons and inconsistent use of quotes on snippets that featured strings. They stated a violation percentile value of 82.9 [37].

The general conclusions drawn by these last two motivating studies by Bafatakis, N. *et al.* [13] and Campos, U. *et al*. [37] about Stack Overflow reliability are in concord with the inferences drawn by the earlier reviewed literature. They all indicate that owing to numerous factors, it is unideal to trust Stack Overflow's content.

4 Methodology

4.1 The Dataset

We used the SOTorrent Dataset for our experiments. The Dataset derives from the official Stack Overflow data dump, built to analyse Stack Overflow content evolution [18]. The SOTorrent datasets are available online via Google BigQuery [44].

4.2 Stack Overflow Content Harvesting

Using Google BigQuery, we pulled 8,998,890 results associated with the 'javascript' tag on Stack Overflow. The CSV file featured a header row that corresponds to the selected fields in the BigQuery query, and these included the following fields postID (as postIdFinal), SpostId (as verifyIdFromVotes), Title, Content, OwnerUserId, LastEditorUserId, ViewCount, AnswerCount, CommentCount, userID, userReputation, UpVotes, DownVotes.

4.3 Code Snippet Development

The next stage involved using a Java program to clean the snippets. This step entailed replacing the SOTorrent-specific CSV delimiter: '`
`' with the usual newline character: '\n' and reconstructing the code snippets from the results (rows) in the CSV files. This stage essentially transformed the content of the 'Content' field of every result (CSV row) into a source code file. The file name pattern is of the form: *Post ID* + "-" + *Post Revision number* + the file extension associated with the subject language, that is, either '*.js*' or '*.py*'. After that, we placed these developed code snippets in a directory. The directory contained subfolders holding around 200,000 source code files each.

4.4 Code Snippet Filtration

We wanted to ensure that every code snippet considered in the experiment duly represented Javascript code. To this end, we used Bash Shell scripting to established three source file filtration criteria. These were source code snippets that contain keywords that are core to the subject language. We considered code snippets that contain the keywords 'let' or 'var' or 'const'; source code snippets that contain characters: '(' or '=' and source code snippets that contain at least six (6) lines of code.

Our Bash script then wrote all the filenames for the code snippets that meet the filtration criteria to a .log file. We then employed this log file as a verifier for qualified snippets during the linting process. We did this by loading the contents of the log file into an array. For every linting process, we would check whether the subject snippet's filename existed in the array. Figure 2: Snippet Figures shows the complete statistics before and after filtration.

4.5 Style Checking

4.5.1 The ESLint Report. To analyse the programming style, we employed ESLint. ESLint is a Node.js driven popular linter for JavaScript dating created in 2013 [40]. ESLint makes it possible to discover issues in JavaScript code without executing it. The tool's goal is to allow software developers to establish their own linting rules.

We developed a linting program in Node.js that leverages the Node.js API [41].; this allows us to use ESLint (which is traditionally a command-line tool) programmatically. Upon execution, the application asks for the absolute path to the log file containing the filenames for the candidate code snippets and the absolute path to the directory holding the JavaScript code snippets. The program then locates the candidate files in the code snippet directories. It performs linting on them, applying JavaScript Programming Style rules as stipulated by the JavaScript Standard Style [42]. The program then formats the result for each snippet. It outputs them to separate JSON files whereby the filename matches the name of the corresponding snippet. For example, the ESLint report for a snippet with the filename 186024-0.js would be 186024-0.json. The tool offers several format options, including 'checkstyle', 'codeframe', 'compact', 'html', 'jslint-xml', 'json', 'junit', 'stylish' (the default), 'table', 'tap', 'unix', 'visualstudio' but we opted for a JSON output.

Our linting program's JSON reports contain the details of the linting process for each code snippet.

We then developed another Java program that we used to parse and analyse the JSON reports. This exercise yielded various insights, notably the most and the least violated rules – see research question 1 and 2. We were also able to identify the code snippets fully compliant with programming style rules based on our custom criteria. Using the filenames of these code snippets, we revisited the results from our Google BigQuery query results to identify these snippets' authors – see research question 3 and 4.

4.5.2 Our Approach.

4.5.2.1 Research Question 1. We iterated through all errors of all ESLint report files. We excluded the files that we could not open for parsing and the unparsable files due to a parsing error.

Also, we excluded ESLint errors [43] that are not errors typically encountered in vanilla (traditional) JavaScript. For example: 'valid-typeof', 'no-unreachable', 'keyword-spacing' are errors that are present in conventional JavaScript, so we considered these types of errors, while we ignored errors such as 'react/jsx-filename-extension', 'node/handle-callback-err' and '@typescript-eslint/no-var-requires' which are errors related to JavaScript, framework/platforms like React.js, Node.js and Typescript respectively. Additionally, we also ignored trivial, easily fixable errors. These include indentations, no tabs and all other styling errors. We decided to exclude these errors because these styling errors' relative triviality meant that they were so prevalent in the snippets. This high volume would have prevented us from achieving meaningful results.

Additionally, another reason for exclusion was that the guidelines under the category are highly subjective. Even the developers of ESLint share this sentiment - as per the ESLint List of Available Rules webpage. This exclusion meant we deemed a report file compliant if all the errors were entire of a stylistic nature. This criterion qualified the corresponding snippet for the ESLint report file as 'compliant'.

After that, we obtained our violation ratio as thus: **the total number of compliant snippets/(total number of parsable ESLint reports – Ignored snippets)**.

4.5.2.2 Research Question 2. We built a map data structure that held every single error encountered in every ESLint report as a key and the number of occurrences of each of these errors as the value. Afterwards, we sorted (highest to lowest) the map by value and outputted the result to a text file. We then transferred this data to a Microsoft Excel sheet for visual inspection and thoroughly applied filters to gain various insights.

4.5.2.3 Research Question 3. We employed two map data structures. The first one collected the post ID of all compliant posts and mapped it to a default Boolean of value 'false'. The second one mapped these post IDs to the file path of the snippets. We iterated through our Google BigQuery query results. For each row, we calculated the violations-per-statement value for each snippet by dividing the total number of errors found in each snippet by the number of statements. We obtained this denominator (number of statements) by dynamically counting the number of statements in each snippet as we looped through our query results. We stored the post IDs for the compliant snippets and the associated aggregate vote in another map in this same loop. We obtained the aggregate vote by calculating the difference between the positive and negative votes for a given post. If the resulting value is a positive number, we conclude that it is a positively received post. We then switch the Boolean value to 'true' in our first map for the associated post ID. We stored the violations-per-statement for each post, mapped to its discrete post score and used these mappings to plot a scatterplot in Figure 8: Violations-per-statement vs Post Score.

We carried out additional microscopic observation under this research question. We grouped posts by their post score. We achieved the grouping by counting the total number of posts with a score of -10 up to score 20. For each post score, we evaluated the mean and median of their violations-per-statement. We then used this information to plot the scatter plot in Figure 9: Mean and Median Values by Post Score groups. We employed the Pearson Correlation Coefficient function from The Apache Commons Mathematics Library. It is a statistical tool used in verifying the existence of a linear relationship between two sets of data. We ran this function on the mean values of violations-per-statement, alongside the corresponding post score. Additionally, We also ran it on the median values of violations-per-statement, alongside the corresponding post score.

4.5.2.4 Research Question 4. After collecting all the post IDs for the qualifying posts, we mapped these IDs to the number of errors in their associated ESLint reports, excluding the ignorable errors. We then created an '`author`' Java class that represented Stack Overflow users. This class features attributes that stored user details. These attributes include user ID, the total number of errors, the user's reputation, the total number of posts. We concentrated on non-collaborative posts for each user, which the original post author last edited.

Furthermore, we targeted users that have made at least five posts to help obtain better average values. Afterwards, we then computed the average violations-per-statement. We calculated this as thus: **(total errors by a user in all snippets they have authored - ignored snippets) /the total number of statements in their snippets in all snippets they have written**. We then plotted the values on a scatterplot for trend observation – see Figure 11: Average Violations-per-statement vs User Reputation. To verify some of the outliers, for example, users with very high reputation values, we implemented a sorted map containing user IDs and associated reputation. We then went over to Stack Exchange Data Explorer [12] to manually run verification queries.

Like research question 3, we also employed the Pearson Correlation Coefficient to verify the existence of a linear relationship between the average number of violations-per-statement by each user and their reputation. We also ran this function on both the mean and median values of the average number of violations-per-statement by each user against their stack overflow reputation.

4.5.2.5 Research Question 5. For our security analysis, we set up three security-related ESLint plugins. These are eslint-plugin-scanjs-rules [48], eslint-plugin-no-unsanitized [46], eslint-plugin-prototype-pollution-security-rules [47].

Afterwards, we declared ESLint rule ids for the security violations we intend to analyse in our configuration file (*.eslintrc.json*). However, we only declared rules from the eslint-plugin-no-unsanitized plugin. We made this decision because after initially declaring all the rule ids for all the plugins, every eligible snippet reported an error. This scenario described above is very similar to our experience with the conventional error check, where almost every code snippet returned an error. Afterwards, we collected all security errors and mapped them to their number of occurrences.

Another reason for narrowing down our list of security rules was because we wanted to ensure that we catered only to vanilla JavaScript. After our first security analysis with all the rule IDs for all the plugins available, we noticed many of the errors related to JavaScript-derived frameworks. This observation held for mostly the Node.js platform and React.js.

We assessed the security violation ratio and obtained the value using this equation: the t**otal number of vulnerable snippets files/total number of eligible snippets**. We also evaluated the security violation-to-total violation ratio by applying the following equation: **the total number of vulnerable snippets/ (the total number of eligible snippets - total number of compliant snippets)**.

5 Results

5.1 Research Question 1

Do JavaScript code snippets on Stack Overflow usually conform to programming style guides?

Pre-Filtration	8,998,890
Post-Filtration	5,300,744
Total Number of Generated Reports	2,607, 995
Readable by Linter	2,593,627
Parsing Error	1,357,823
Eligible Snippets	1,235,804
Compliant Snippets	64,499
Violation Ratio	0.95

Figure 2: Snippet Figures

5.1.1 Discussion (RQ1). After filtration, ESLint was able to generate a total of 2,607,995 reports. Out of these, our JSON parser was able to parse 2,593,627 files. One million three hundred fifty-seven thousand eight hundred twenty-three (1,357,823) could not be parsed entirely due to parsing errors. We then had a remainder of 1,235,804 eligible snippets for further processing and analysis. This figure was the basis of the rest of our experiment, especially the violation ratio equations.

We analysed these remaining files for errors. As explained earlier, we initially excluded only errors relating to improper indentation, inconsistent use of semicolons, tabs and inconsistent use of double and single quotes. But, we found that the violation rate was still very high. We then decided to exclude the entire 'Stylistic Issues' category – see ESLint List of Available Rules [43]. This exclusion reduced our main error violation ratio from almost '1.0' to '0.95'. It also increased the compliant posts from 107 to 64,499.

The tabulated violation ratio of **0.95** thus indicates that developers cannot trust JavaScript code on Stack Overflow. These code snippets cannot be considered model code for learning JavaScript.

In comparison to the Python paper by Bafatakis, N. *et al*. [13], Python's violation ratio was 0.7. This figure implies that Python snippets may be more reliable than their JavaScript counterpart.

5.2 Research Question 2

Which is the most violated JavaScript programming guidelines on Stack Overflow?

Error	Count	Absolute Rank
no-console	1,046,861	6
no-obj-calls	949,930	8
no-ex-assign	394,387	10
no-useless-backreference	257,317	13
no-sparse-arrays	86,578	20

Figure 3: Possible Errors - Top Five Errors

Error	Count	Absolute Rank
no-case-declarations	983,654	7
eqeqeq	362,197	11
no-empty-pattern	265,905	12
no-multi-spaces	203,559	16
dot-notation	85,006	21

Figure 4: Best Practices - Top Five Errors

Error	Count	Absolute Rank
no-undef	5,081,715	1
no-unused-vars	636,496	9
no-undef-init	20,647	35
no-use-before-define	11,616	43
no-shadow-restricted-names	952	86

Figure 5: Variables - Top Five Errors

Error	Count	Absolute Rank
no-var	2,467,874	2
no-dupe-class-members	1,839,491	3
no-this-before-super	1,626,874	4
constructor-super	1,626,756	5
arrow-spacing	220,697	14

Figure 6: ECMAScript 6 - Top Five Errors

Category	Number of Distinct Errors
Best Practices	43
Possible Errors	32
ECMAScript 6	17
Variables	5

Figure 7: Distinct Errors Grouped By ESLint Error Categories

5.2.1 Discussion (RQ2). As seen from the ESLint List of Available Rules web page [43], there are eight (8) categories of errors: Possible Errors, Best Practices, Strict Mode, Variables,

Stylistic Issues, ECMAScript 6, Deprecated and Removed. The tables above present the top five errors for each category found in the snippets after eliminating the snippets that do not qualify based on our ignorable errors. We found **97** distinct errors with total occurrences of **19,931,589** that span a total of four categories, including Possible Errors, Best Practices, Variables, and ECMAScript 6. Additionally, we also observed that Possible Errors and the Best Practices were the categories where most errors fell. We also observed that a total of **264,514** errors were uncategorised by ESLint. More information on the descriptions of the reported errors is on the ESLint webpage [43].

The findings tabulated in Figure 7: Distinct Errors Grouped By ESLint Error Categories further clarifies that Stack Overflow is not a reliable source for model JavaScript code. As can be observed, the platform's most flouted rules are the essential guidelines basic to JavaScript programming. The ESLint List of Available Rules webpage [43] describes their 'Best Practices' category as the rules concerned with better methods of doing things to mitigate possible issues. Additionally, they also define their 'Possible Errors' class as those types of errors that relate to syntax and logic. As one can see, it is not difficult to deduce that these descriptions cater to the essential guidelines that a language can stipulate. But, at the same, the table in Figure 7: Distinct Errors Grouped By ESLint Error Categories shows that these basic rules are by far the most flouted.

On the other hand, Bafatakis, N. *et al*. [13] found that out of their 5,076,647 errors, a third of them related to improper use of whitespace. They also found that the top ten violated rules accounted for about 85% of the total violations. Finally, they grouped the errors under their relevant category. They found out that most errors were under the 'Convention' category, followed by the 'Warning and 'Refactor' and lastly, the 'Errors' category.

5.3 Research Question 3

Are compliant JavaScript posts consistently positively voted?

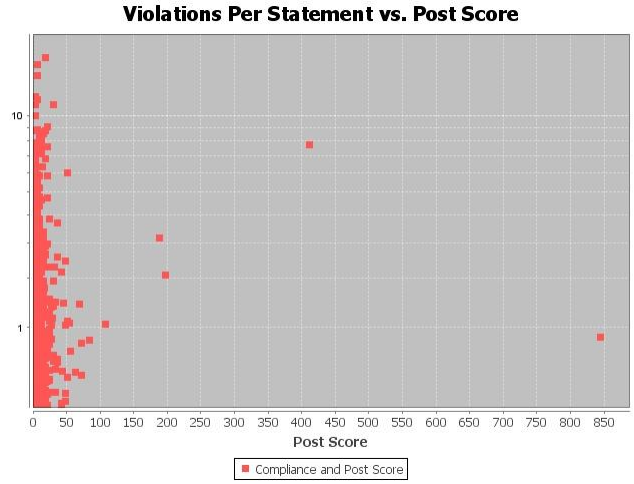

Figure 8: Violations-per-statement vs Post Score

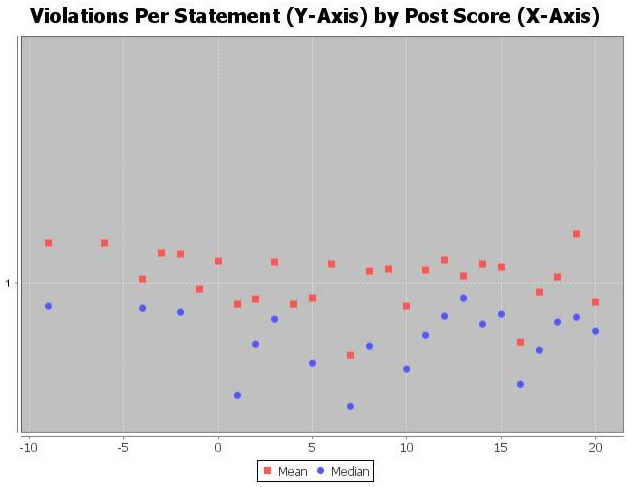

Figure 9: Mean and Median Values by Post Score groups

Range	Mean			Median	
	R-Value	P-Value		R-Value	P-Value
[-10, +20]	-0.2362	.235979		-0.1613	.422409
[-10, +25]	-0.2198	.212381		-0.1301	.337927
[-10, +30]	-0.1822	.094391		-0.1095	.359047
[-10, +35]	-0.1442	.141587		-0.0845	.168963
[-10, +40]	-0.1109	.106190		-0.0612	.147843

Figure 10: Pearson Correlation Coefficient Values (*I*)

5.3.1 Discussion (RQ3). Observing the chart in Figure 8: Violations-per-statement vs Post Score exhibits a very weak relationship between violations-per-statement and post score. However, it is somewhat noticeable that posts with a relatively high score (values 40-100 on the axis) start to diminish as the violations-per-statement rises. But, there were a few outliers with relatively high post scores and equally high violations-per-statement values.

Additionally, we computed the ratio of positively voted compliant posts to the total number of compliant posts to be **0.31**. This ratio further supports the trend observed in the chart in Figure 8: Violations-per-statement vs Post Score. It confirms that most posts that conform to programming guidelines do not necessarily turn out to be well-received posts. Moreover, there still exists a weak inverse proportion trend between violation-per-statements and post scores. This phenomenon suggests that positive votes may be a false measure for the quality of JavaScript snippets on Stack Overflow.

On the other hand, for Python snippets, the violation-per-statement exhibited a stronger inversely proportional relationship to post scores, essentially forming a Zipf distribution.

Like Bafatakis, N. *et al*. [13], we opted to perform a better microscopic observation. To that end, we evaluated the mean and median values of the violations-per-statement values of all post score groups (above 50 counts) between -10 and 20. We plotted a chart with these values accordingly. And like Bafatakis, N. *et al*. [13], after mapping post scores to their frequencies, we also noticed that the stated range, that is, -10 to 20, also housed a

majority of our eligible snippets. Figure 9: Mean and Median Values by Post Score groups. shows the plot.

The R-value of Pearson's Correlation Coefficient for the mean is **-0.2362** for our primary range of [-10 to 20], suggesting a negative and weak relationship. Additionally, the P-value for this range is **.235979**, an insignificant result. The median follows the same trend as the R and P values are **-0.1613** and **.422409**, respectively. Figure 10: Pearson Correlation Coefficient Values shows the R- and P values for different post score ranges. We observed that the R values diminish as we go down the table rows. This phenomenon indicates that the relationship gets even weaker the more we widen the ranges.

5.4 Research Question 4

Do highly reputed users follow programming style guides more than lowly reputed users?

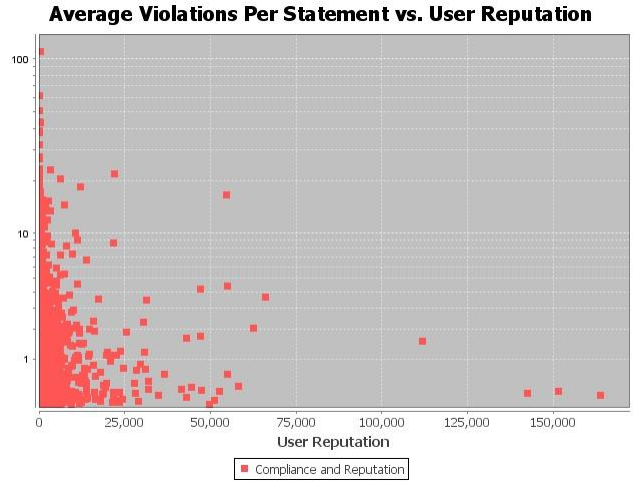

Figure 11: Average Violations-per-statement vs User Reputation

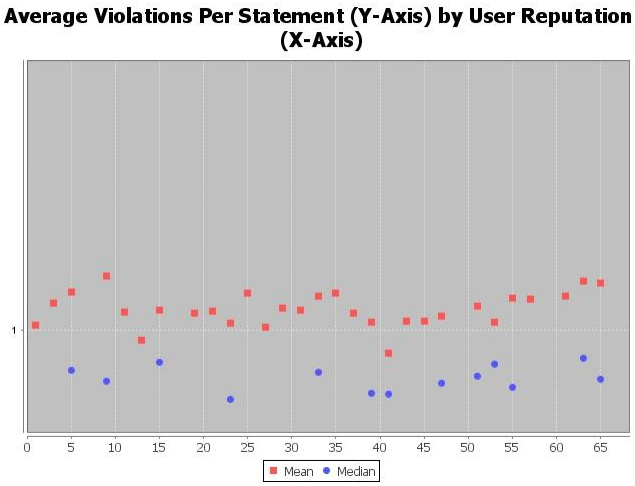

Figure 12: Mean and Median Values by User Reputation

Range	Mean			Median	
	R-Value	P-Value		R-Value	P-Value
[-10, +20]	0.1718	.372861		0.0743	.701684
[-10, +25]	0.1556	.354217		0.0662	.421019
[-10, +30]	0.1421	.298288		0.0230	.491178
[-10, +35]	0.1210	.149144		0.0325	.280673
[-10, +40]	0.1001	.223716		0.0187	.385962

Figure 13: Pearson Correlation Coefficient Values (*II*)

5.4.1 Discussion (RQ4). In a similar fashion to research question 3, we charted the mean and median values of average violation per statement values against user reputation. However, to limit the values to a reasonable output, we calculated our mean and median values on groups of at least 500 instead of 50, as used in research question 3. Figure 11: Average Violations-per-statement vs User Reputation suggests a weak correlation between the average violations-per-statement and a user's reputation. However, as indicated by the positive R-value, the correlation is of a positive variety. For the mean, the R-value for our primary range is **0.1718,** and the corresponding P-value is an insignificant value of **.372861**. And as for the median, the R and P values are **0.0743** and **.701684**, respectively. Similar to research question 3, Figure 13: Pearson Correlation Coefficient Values (*II*) also shows a table of Pearson's Correlation Coeefieinct values with varying ranges. We also see the R-values go down as the ranges widen. These findings are similar to Python's conclusions. They discovered no correlation between average violations-per-statement and user reputation.

5.5 Research Question 5

Do JavaScript code snippets on Stack Overflow usually conform to secure programming standards?

Security Rule ID (Error)	Count
no-unsanitized/property	8,763,001
no-unsanitized/method	5,512,038

Figure 14: Security Issues

Security Violations	14,275,039
Vulnerable Snippets	921,270
Security Violation Ratio	0.75
Security Violation-to-Error Violation Ratio	0.78

Figure 15: Security Issues Statistics

5.5.1 Discussion (RQ5). 'no-unsanitised/property' refers to a basic security check for property assignments to ensure secure assignment operations [46]. The 'innerHTML' feature is convenient because it allows a developer to replace content on a webpage easily. However, this attribute makes innerHTML susceptible to cross-site scripting attacks as it may enable an attacker to inject malignant text into HTML tags [49]. This rule helps mitigate that risk.

The second rule, 'no-unsanitised/method,' forbids certain function calls. For example, '`document.write()`'. Like the previous example, an attacker can exploit this feature for cross-site scripting attack by using it to surreptitiously steal browser cookies or modify a web page's behaviour [50].

6 Research Contributions

To the best of our knowledge, many Stack Overflow code compliance studies consider just one programming language. However, we adopted a point-to-point comparative approach in this study by replicating similar research on Python. This approach enabled us to intimately compare the findings across JavaScript and Python as the experiment progressed. This comparison highlighted the differences between the languages and painted a clearer picture of the characteristics that different languages could embody even on the same platform.

Notably, this benchmarking approach revealed the varying nature of errors across JavaScript and Python. It also provided insight into how the two languages correlate with the Stack Overflow system. That is, compliance-to-votes and compliance-to-user reputation. For instance, while compliant Python posts have a strong relationship with positive votes, JavaScript does not show the same trend.

In addition to conventional rule violations, our work also explored the security of these JavaScript snippets. Our analysis provided some insights into the vulnerability levels of these code snippets.

7 Study Limitations

The main limitation was that it was impossible to create reports for the entire 8,998,890 JavaScript snippets found. This phenomenon happened for various reasons—some of the core reasons being parsing issues and partial code. Also, the exclusion of trivial errors meant that we could not get the entire picture. However, it was a worthy path to ensure that the research focused only on the priority errors while eliminating trivial and subjective errors that barred us from attaining meaningful results.

8 Threats to Validity

8.1 Internal Validity

8.1.1 Collection of Non-collaborative Posts. The database schema features the following fields:

- `Field{name=OwnerUserId, type=INTEGER, mode=NULLABLE, description=null, policyTags=null}`
- `Field{name=LastEditorUserId, type=INTEGER, mode=NULLABLE, description=null, policyTags=null}`

The first one holds the author's user ID of a post, and the second field holds the ID of the last editor of a post. As explained earlier, as part of RQ4, to identify non-collaborative posts we filtered for posts written by one author by using the condition: `if (ownerUserId == lastEditorUserId && ownerUserId != -1) {…}`. This if-statement will identify posts that satisfy the specified condition. However, posts edited by other users between post creation and final editing by the same post creator will slip through the if-statement. This situation creates a flaw in the logic.

8.1.2 Collection of Non-collaborative Posts. As per Section 4.4, to ensure that we consider snippets representing JavaScript code, we filtered out snippets with fewer than six lines. However, given Stack Overflow's nature as a community question answering platform, it is usual practice for users to answer a question with incomplete code. For instance, if an enquirer asks how to obtain the current year in JavaScript, it would be sufficient to respond to the query by providing an answer like this: '`let currentYear = new Date().getFullYear();`'. While this answer represents working JavaScript code and may satisfy the enquiry, our criteria would still filter it out. Furthermore, our standards were stipulated independently, based on what we deemed adequate for obtaining sensible results. Other similar studies that have similar snippet filtration mechanisms will almost certainly do things differently, too.

8.1.3 Ignored Errors. Upon our initial analysis run, every single snippet contained an error. Most of these errors were related to trivial styling, which can be very subjective. To achieve workable results and identify compliant snippets, we narrowed our scope. We did that by focusing on the core errors related to JavaScript's logic and syntax. This way, a snippet that contains mistakes entirely made up of our ignored category (related to trivial styling) was considered compliant. This decision, of course, means that the results will differ from other similar studies.

8.1.4 Javascript Code Snippet Retrieval. In our original Google BigQuery, which pulled the Dataset used in this study, we specified the condition: `WHERE posts.Tags LIKE '%javascript%'`. This approach is not airtight, mainly because of question mistagging. Mistagging is a common occurrence on Stack Overflow, especially amongst new programmers and new platform users. So there is a possibility that the condition will not return all JavaScript-related materials. We observed this on a milder note when working with the snippets. Some of the errors we found were not related to vanilla JavaScript. Instead, they related to frameworks like Angular, React.js,Vue.js, and Node.js.

8.1.5 ESLint. Out of all the snippets that passed our filtration criteria described in Section 4.4, ESLint did not create JSON reports for all of them. Additionally, on subsequent runs, ESLint could not read a relatively small number of the reports too. Furthermore, our JSON parser could not parse a significant number of snippets due to parsing errors. These technical obstacles translated into a significantly lower number of eligible

snippets qualified for further and final processing. As a result, the findings would differ for other researchers depending on their setups. More details about the figures are available in Section 5.1.

8.1.6 The Dataset. The SOTorrent Dataset used for the study is one of several versions. The data on Stack Overflow is continuously changing. This rapid change is because of its status as one of the modern websites with very high traffic. As a result, subsequent similar studies on JavaScript on Stack Overflow could yield vastly different results.

8.2 External Validity

This study centres on traditional JavaScript with intimate comparisons to another closely-related research on Python. So the details in this study only relate to the two languages and their use on Stack Overflow. Furthermore, JavaScript code obtained from other sources other community question answering platforms other than Stack Overflow may show different findings.

9 Related Work

As indicated earlier, one of the inspiring papers that prompted this work was by Ferreira Campos, U. et al. [37]. Like theirs, our study employed both ESLint and SOTorrent. However, they experimented on a much smaller number of snippets than our research at 336K. Regarding the findings, similar to our study, they also found no one code snippet without a violation. This finding held before we excluded the Stylistic Issues category. Additionally, they also found out that stylistic issues accounted for most of the errors by a wide margin. However, the proportions of the mistakes found under the 'Possible Errors' categories seemed to differ significantly between our studies.

In 2016, Di Yang, Aftab Hussain, and Cristina Videira Lopes [45] carried out extensive usability analysis on three million snippets spanning a few programming languages used on Stack Overflow. These languages included C#, Java, JavaScript and Python. Interestingly they reported Python and JavaScript as the languages with the most usable snippets on the platform. Their findings directly contrast with ours and Ferreira Campos, U. et al.'s [37] findings for JavaScript. Additionally, the same contrast level is observable when comparing their results with Bafatakis, N. et al. [13] for Python.

On the other hand, they concluded that the dynamic languages' usability is much higher than that of the statically typed languages. The dynamic languages refer to Python and JavaScript, and the static languages refer to C# and Java. This conclusion agrees with our choice of language for this study, as explained in Section 2.4. We expressed that the ability to assess partial code lines was a benefit that JavaScript offers for our research.

10 Future Work

In the future, we would be interested in expanding this study to cover not only vanilla JavaScript but its core frameworks, including Angular, Node.js and Vue.js. A full-scale linting analysis that encompasses these frameworks and environments will help paint a richer picture of the JavaScript language's compliance levels on Stack Overflow.

Additionally, we would also like to evaluate the correlation between response times and their correlation to a post's violation per statement. This analysis will provide insight into the gamification of Stack Overflow.

11 Conclusion

After our data analysis, we found that JavaScript code on Stack Overflow has an error violation ratio of **0.95** and a security violation ratio of **0.75**. This value is higher than what was evaluated for Python by Bafatakis, N. *et al*. [13].

We also found that most flouted rules were the most crucial guidelines. To be more specific, we found a total of **97** distinct errors. Out of which **43** (~ 44.3%) unique errors belonged to the 'Best Practices' category, a category that recommends guidelines for doing things to avoid issues such as latent bugs and unexpected results. **32** (~ 33%) distinct errors were under the 'Possible Errors' category, a crucial category that stipulates guidelines that could help a JavaScript programmer avoid syntactic and logic errors. **17** (~17.5%) distinct errors came under the ECMAScript 6 category, which relates to guidelines that enforce the 'modern ways' of writing JavaScript code based on the ECMAScript 6 JavaScript revision in 2015. This version introduced keywords such as 'let' and 'const'. Finally, the **5** (~5.15%) last distinct errors belonged to the 'Variables' category concerned with JavaScript variable declarations.

We further explored the Stack Overflow (scoring) system and its possible influences and relationship with developers' code contributions. We found a very weak correlation between compliance post score. We also found that the ratio of positively-received compliant snippets to the total number of compliant snippets stood at **0.31**. Additionally, we discovered a weak relationship between a user's average violations per statement and their reputation.

We advise that developers apply caution and justifiable mistrust when working with JavaScript code available on Stack Overflow. This conclusion secondarily suggests that the idea of blindly copying and reusing JavaScript code from Stack Overflow could have detrimental effects on the future evolution of recipient software artefacts.

REFERENCES

[1] Lim, J. et al. 2011. Style Avatar. *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11*. (2011).

[2] Li, X. and Prasad, C. 2005. Effectively teaching coding standards in programming. *Proceedings of the 6th conference on Information technology education - SIGITE '05*. (2005).

[3] Spencer, H. et al. 1997. *Recommended C Style and Coding Standards*. Bell Labs.

[4] PEP 8 -- Style Guide for Python Code: 2020. *https://www.python.org/dev/peps/pep-0008/*. Accessed: 2020- 07- 10.

[5] Standard ECMA-262: 2020. *https://www.ecma-international.org/publications/standards/Ecma-262.htm*. Accessed: 2020- 07- 10.

[6] JavaScript: 2020. *https://developer.mozilla.org/en-US/docs/Web/JavaScript*. Accessed: 2020- 07- 10.

[7] Programming Style | CS Student Handbook: 2020. *https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/programming-style.html*. Accessed: 2020- 07- 10.

[8] Stack Overflow - Where Developers Learn, Share, & Build Careers: 2020. *https://stackoverflow.com/*. Accessed: 2020- 07- 13.

[9] Stack Overflow: 2020. *https://insights.stackoverflow.com/trends?tags=*. Accessed: 2020- 07- 13.

[10] JavaScript | TIOBE - The Software Quality Company: 2020. *https://www.tiobe.com/tiobe-index/javascript/*. Accessed: 2020- 07- 13.

[11] All Sites - Stack Exchange: 2020. *https://stackexchange.com/sites?view=list#users*. Accessed: 2020- 07- 13.

[12] Stack Exchange Data Explorer: 2020. *https://data.stackexchange.com/*. Accessed: 2020- 07- 13.

[13] Bafatakis, N. et al. 2019. Python Coding Style Compliance on Stack Overflow. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. (2019).

[14] Python | TIOBE - The Software Quality Company: 2020. *https://www.tiobe.com/tiobe-index/python/*. Accessed: 2020- 07- 14.

[15] index | TIOBE - The Software Quality Company: 2020. *https://www.tiobe.com/tiobe-index/*. Accessed: 2020- 07- 14.

[16] Ahmad, M. and Cinneide, M. 2019. Impact of Stack Overflow Code Snippets on Software Cohesion: A Preliminary Study. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. (2019).

[17] Acar, Y. et al. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. *2016 IEEE Symposium on Security and Privacy (SP)*. (2016).

[18] Baltes, S. et al. 2018. SOTorrent. *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18*. (2018).

[19] Srba, I. and Bielikova, M. 2016. Why is Stack Overflow Failing? Preserving Sustainability in Community Question Answering. *IEEE Software*. 33, 4 (2016), 80-89.

[20] How To Keep Stack Overflow from Turning Into a Landfill: 2020. *https://www.vice.com/en_us/article/9a3q8z/whither-stack-overflow*. Accessed: 2020- 07- 14.

[21] An, L. et al. 2017. Stack Overflow: A code laundering platform?. *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. (2017).

[22] Welcome to Python.org: 2020. *https://www.python.org/*. Accessed: 2020- 07- 14.

[23] All Sites - Stack Exchange: 2020. *https://stackexchange.com/sites#traffic*. Accessed: 2020- 07- 15.

[24] Lopez, T. et al. 2018. An investigation of security conversations in stack overflow. *Proceedings of the 1st International Workshop on Security Awareness from Design to Deployment - SEAD '18*. (2018).

[25] Wiese, E. et al. 2017. Teaching Students to Recognise and Implement Good Coding Style. *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale - L@S '17*. (2017).

[26] VanNeste, K. 1986. Ada coding standards and conventions. *ACM SIGAda Ada Letters*. VI, 1 (1986), 41-48.

[27] Kernighan, B. and Plauger, P. 1974. Programming Style: Examples and Counterexamples. *ACM Computing Surveys (CSUR)*. 6, 4 (1974), 303-319.

[28] Abdalkareem, R. et al. 2017. What Do Developers Use the Crowd For? A Study Using Stack Overflow. *IEEE Software*. 34, 2 (2017), 53-60.

[29] Doerner, C. et al. 2014. EUKLAS. *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools - PLATEAU '14*. (2014).

[30] Ginsca, A. and Popescu, A. 2013. User profiling for answer quality assessment in Q&A communities. *Proceedings of the 2103 workshop on Data-driven user behavioral modelling and mining from social media - DUBMOD '13*. (2013).

[31] Jin, Y. et al. 2015. Quick Trigger on Stack Overflow: A Study of Gamification-Influenced Member Tendencies. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. (2015).

[32] Anand, D. and Ravichandran, S. 2015. Investigations into the Goodness of Posts in Q&A Forums—Popularity Versus Quality. *Advances in Intelligent Systems and Computing*. (2015), 639-647.

[33] Asaduzzaman, M. et al. 2013. Answering questions about unanswered questions of Stack Overflow. *2013 10th Working Conference on Mining Software Repositories (MSR)*. (2013).

[34] Ragkhitwetsagul, C. et al. 2019. Toxic Code Snippets on Stack Overflow. *IEEE Transactions on Software Engineering*. (2019), 1-1.

[35] Shannon, B. 1993. *C Style and Coding Standards for SunOS*. Sun Microsystems.

[36] C style for your source code: 2020. *https://devnull-cz.github.io/unix-linux-prog-in-c/cstyle.html*. Accessed: 2020- 07- 17.

[37] Ferreira Campos, U. et al. 2019. Mining Rule Violations in JavaScript Code Snippets. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. (2019).

[38] Yang, X. et al. 2016. What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts. *Journal of Computer Science and Technology*. 31, 5 (2016), 910-924.

[39] Stack Exchange Data Dump : Stack Exchange, Inc. : Free Download, Borrow, and Streaming : Internet Archive: 2020. *https://archive.org/details/stackexchange*. Accessed: 2020- 07- 23.

[40] ESLint - Pluggable JavaScript linter. Retrieved January 25, 2021 from https://eslint.org/

[41] Node.js API - ESLint - Pluggable JavaScript linter. Retrieved January 25, 2021 from https://eslint.org/docs/developer-guide/nodejs-api

[42] standard/standard: 🌟 JavaScript Style Guide, with linter & automatic code fixer. Retrieved January 25, 2021 from https://github.com/standard/standard

[43] List of available rules - ESLint - Pluggable JavaScript linter. Retrieved February 23, 2021 from https://eslint.org/docs/rules/

[44] SQL workspace – BigQuery – Google Cloud Platform. Retrieved March 4, 2021 from https://console.cloud.google.com/bigquery?project=sotorrent-org&p=sotorrent-org&d=2020_03_15&page=dataset

[45] Di Yang, Aftab Hussain, and Cristina Videira Lopes. 2016. From query to usable code: An analysis of Stack Overflow code snippets. In *Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016*. DOI:https://doi.org/10.1145/2901739.2901767

[46] eslint-plugin-no-unsanitized - npm. Retrieved March 4, 2021 from https://www.npmjs.com/package/eslint-plugin-no-unsanitized

[47] eslint-plugin-prototype-pollution-security-rules - npm. Retrieved March 4, 2021 from https://www.npmjs.com/package/eslint-plugin-prototype-pollution-security-rules

[48] eslint-plugin-scanjs-rules - npm. Retrieved March 4, 2021 from https://www.npmjs.com/package/eslint-plugin-scanjs-rules

[49] innerHTML Cross-site Scripting - DEV Community. Retrieved March 6, 2021 from https://dev.to/caffiendkitten/innerhtml-cross-site-scripting-agc

[50] What is DOM Based XSS (Cross-site Scripting)? | Netsparker. Retrieved March 6, 2021 from https://www.netsparker.com/blog/web-security/dom-based-cross-site-scripting-vulnerability/