ExampleCheck: Assessing Stack Overflow Code Snippets with API Usage Patterns Mined from GitHub

Anonymous Author(s)

ABSTRACT

Programmers often consult Q&A websites such as Stack Overflow to learn new APIs. However, online code snippets are not always complete or reliable in terms of API usage. To help programmers assess online code snippets, we build a Chrome-extension, ExampleCheck that detects API usage violations in Stack Overflow posts using patterns mined from 380K GitHub projects. ExampleCheck quantifies how many GitHub examples follow the correct API usage and also provides the remediation of a detected violation. With the assistance of ExampleCheck, programmers can easily identify the limitations or pitfalls in a Stack Overflow snippet without cross-checking multiple posts for proper API usage reference. They can also build confidence on a code snippet by learning how much it is aligned with other similar code in GitHub. ExampleCheck is available at Chrome Web Store[1] and the demo video is available at https://youtu.be/X0WIZBa8ETA.

CCS CONCEPTS

• **Software and its engineering** → *Software reliability; Collaboration in software development; Integrated and visual development environments;*

KEYWORDS

online Q&A forum, API usage pattern, code assessment

ACM Reference Format:

Anonymous Author(s). 2018. ExampleCheck: Assessing Stack Overflow Code Snippets with API Usage Patterns Mined from GitHub. In *Proceedings of The 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Programmers often search for online code examples to learn new APIs. A case study at Google shows that developers issue an average of 12 code search queries per weekday [10]. Stack Overflow (SO) is a popular Q&A website that programmers often consult. In July 2017, Stack Overflow has accumulated more than 22 million answers, many of which contain code snippets to demonstrate solutions for specific programming questions. However, SO snippets are not always complete or reliable, which can be misleading and sometimes harmful when programmers follow them *as-is* in software development. For example, Fischer et al. found that 29% of security-related code snippets in Stack Overflow were insecure and might affect over 1 million Android apps in Google play [6].

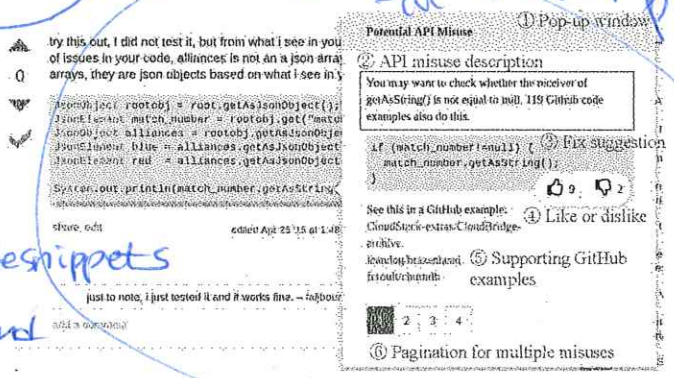

Figure 1: The ExampleCheck Chrome extension that augments Stack Overflow with API misuse warning. The pop-up window alerts that `match_number` can be `null` if the requested `JSON` attribute does not exist and will crash the program by throwing `NullPointerException` when `getAsString` is called on it.

This paper presents ExampleCheck, a Chrome extension that informs programmers about API usage violations in Stack Overflow posts. Figure 1 shows a screenshot of ExampleCheck. To detect API usage violations, ExampleCheck contrasts SO snippets with API usage patterns mined from 380K GitHub repositories in a previous study [16]. These patterns abstract away syntactic details such as variable names, but retain the temporal ordering, control structures, and guard conditions of API calls. Our insight is that common API usage from a large corpus may represent a desirable pattern that a programmer can use to trust and enhance online code snippets.

Given a SO post, ExampleCheck first extracts the sequence of API calls with corresponding control constructs and guard conditions. ExampleCheck then contrasts the call sequence with corresponding patterns learned from GitHub and highlights method calls that violate the common usage patterns. To help users better understand a detected violation, ExampleCheck generates a descriptive warning message and also provides a remediation that fixes the violation. To help developers build confidence on a detected violation, ExampleCheck quantifies how many GitHub developers also follow the same pattern and provides a sample of supporting GitHub examples. However, using common API usage patterns to detect violations may lead to false alarms, since these patterns do not represent infrequent but correct API usage scenarios. To mitigate this issue, ExampleCheck allows users to upvote or downvote a violation based on its applicability and usefulness to a SO post.

[1] Download the Chrome extension of ExampleCheck at https://chrome.google.com/webstore/detail/examplecheck/amliempebckaiaklimcpopomlnklkioe

ESEC/FSE 2018, 4–9 November, 2018, Lake Buena Vista, Florida, United States

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

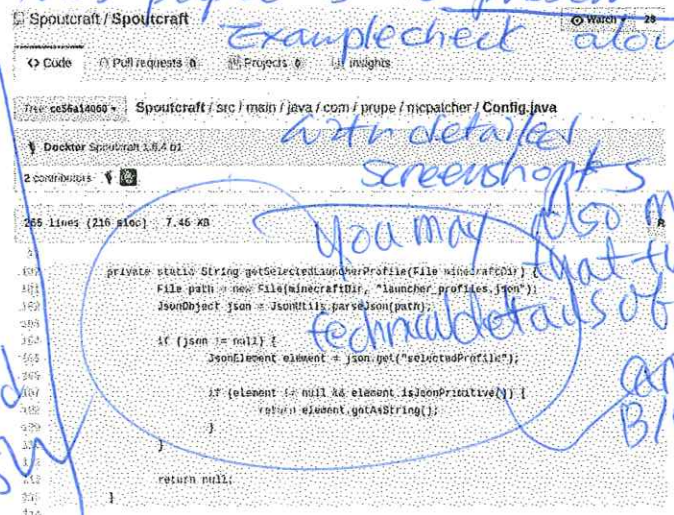

Figure 2: ExampleCheck will redirect the programmer to a concrete code example in GitHub that follows the correct API usage pattern, when she clicks on one of the three GitHub example links in the pop-up window.[2]

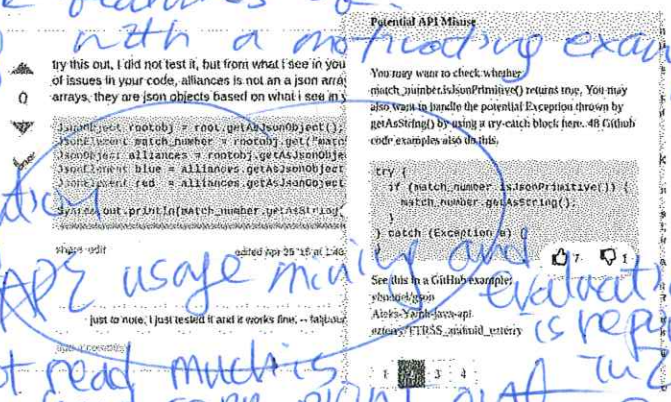

Figure 3: Another API usage warning that reminds programmers to check whether the `JsonElement` object represents a `JSON` primitive value by calling the `isJsonPrimitive` method. It also suggests to catch potential exceptions thrown by `getAsString`.

A user of ExampleCheck would benefit from the addition of concrete examples from the production code in GitHub, when inspecting or reusing code snippets in Stack Overflow. This will not only combat programming issues stemming from the use of incomplete or unreliable SO code snippets, but will also be an aid for users learning a new API. By enhancing examples already found in Stack Overflow, a user can trust that the shown example follows a common and reliable usage pattern for a given API method.

2 AN EXAMPLE AND TOOL FEATURES

Suppose Alice wants to read attribute values from a `JSON` message using Google's Gson library. Alice searches online and finds a related Stack Overflow post with an illustrative code example.[3] Though this post is accepted as the correct answer, it does not properly use the `JsonElement.getAsString` method (line 7 in the snippet in Figure 1), which gets the `string` value of a `JSON` element. For example, if the requested attribute does not exist in the `JSON` message, the preceding API call, `JsonObject.get` (line 2) may return `null`, which consequently leads to `NullPointException` when calling `getAsString` on the returned object. If Alice puts too much trust on this example, she may inadvertently follow a suboptimal solution, which might lead to runtime errors in some edge cases.

Alice cannot easily recognize the potential limitation in the given SO post, unless she manually investigates many other posts and finds code examples that handle such corner cases. In practice, however, programmers often inspect a handful of search results due to the limited time and attention [2, 11]. ExampleCheck frees Alice from this manual investigation labor by contrasting a Stack Overflow post with common API usage patterns learned from over 380K GitHub repositories. ExampleCheck then highlights the API calls that have potential API usage violations in the Stack Overflow post. When Alice clicks on a highlighted API call, ExampleCheck generates a pop-up window with detailed descriptions about the API usage violation, as shown in Figure 1.

API misuse description. To help Alice understand a detected API usage violation, ExampleCheck translates the violation to a short natural language description (see ① in Figure 1). From the short warning message, Alice learns that she should check whether the `JsonElement` object is `null` before calling `getAsString`. Furthermore, ExampleCheck shows that 119 GitHub examples also follow the same API usage pattern. Such quantification can provide additional evidence about how many real-world examples are inconsistent with the given SO snippet and help Alice build confidence on the suggested API usage pattern.

Fix suggestion. ExampleCheck further sketches a fix to demonstrate how to fix the violation in the original SO post, as shown in ③ in Figure 1. This fix is an embodiment of the correct API usage pattern in the context of the SO post. To mitigate the gap between the fix and the original post, we reuse the same variable names in the original SO posts to generate the fix. For example, the `JsonElement` variable in the fix is named as the same variable, `match_number` in the original post.

Demonstrative GitHub examples. To help Alice understand how the same API method is used in real-world projects, ExampleCheck provides three GitHub examples that follow the suggested API usage pattern (see ⑤ in Figure 1). Alice is curious about how others use `JsonElement.getAsString`, so she clicks on the link of the first GitHub example. ExampleCheck redirects Alice to a GitHub page and automatically scrolls down to the Java method where `JsonElement.getAsString` is called, as shown in Figure 2. Compared with the simplified SO example in Figure 1, this GitHub code is more carefully constructed with multiple `if` checks. For example, it not only checks whether the `JsonElement` object is `null`, but also checks whether it is a primitive type to avoid `ClassCastException` before calling `getAsString`. By providing the traceability to concrete code examples in GitHub, Alice could gain a more comprehensive

[2] https://goo.gl/YHo1UM
[3] https://stackoverflow.com/questions/29860000

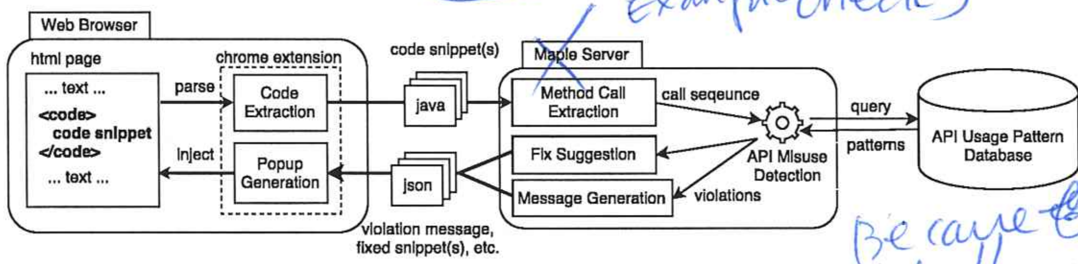

Figure 4: An overview of ExampleCheck's architecture

view of correct API usage in production code, which may not be illustrated in simplified code examples in Stack Overflow.

User feedback. After investigating the concrete example in GitHub, Alice feels that it is necessary to perform the `null` check. She upvotes the pattern by clicking on the "thumbs-up" button to notify other users that this detected violation is helpful to her (see ④ in Figure 1). Alice also finds that her decision resonates with the majority of ExampleCheck users, since nine other users also upvoted this detected violation.

Multiple API usage violations. If a method call in a SO post violates multiple API usage patterns, ExampleCheck ranks these violations by the vote score and the number of supporting GitHub examples and displays them in separate pages in a pop-up window. As shown in ⑥ in Figure 1, the method call, `getAsString` violates four API usage patterns. Figure 3 shows the second violated pattern and suggests Alice to check whether the `JsonElement` object is primitive before calling `getAsString`. Otherwise, `getAsString` will throw `ClassCastException`. ExampleCheck also suggests Alice to wrap `getAsString` with a `try-catch` block to handle potential exceptions. This pattern is supported by 48 GitHub examples.

3 IMPLEMENTATION

This section describes the implementation details of ExampleCheck. Figure 4 shows the architecture of ExampleCheck. When a user loads a Stack Overflow page in the Chrome browser, the Chrome extension extracts code snippets within <code> tags in answer posts, and sends them to the back-end server. The back end then detects API usage violations in a snippet and synthesizes violation descriptions and fixes. For each misused method call in a snippet, the Chrome extension generates a pop-up window using the Bootstrap popover plug-in[4] to inform the user about the API misuse information.

API Usage Pattern Dataset. ExampleCheck leverages 180 patterns of 100 popular Java API methods mined from 380K GitHub repositories in a prior study [16]. These patterns are represented as API call sequences with surrounding control constructs. Each API call is also annotated with its argument types and guard conditions. For example, one pattern, `loop {; get(int)@arg0<rcv.size(); }`, checks if the index is out of bounds when calling the `get` method on an `ArrayList` object. The mining algorithm and pattern format are described in [16]. The current dataset can be extended with API usage patterns learned from other mining techniques [7, 9, 14, 17].

API Misuse Detection. The server first extracts the API call sequence of each SO snippet sent from the front end. We use a partial program analysis and type resolution framework to handle incomplete snippets and resolve ambiguous types [12]. ExampleCheck then queries the pattern database for the API calls present in each API call sequence. Given an API call sequence and an API usage pattern, it checks whether (1) the API calls and control constructs in the sequence follow the same temporal order in the pattern, and (2) the guard condition of each API call in the sequence implies the guard of the corresponding API call in the pattern. ExampleCheck uses a SMT solver, Z3 [4], to check whether one guard condition implies another. For a SO snippet with multiple methods, ExampleCheck inlines the call sequence of an invoked method into the sequence of the caller to emulate a lightweight inter-procedural analysis. ExampleCheck is capable of detecting three types of API usage violations—*missing control constructs*, *missing or incorrect order of API call*, and *incorrect guard condition.*

Warning message generation. Given an API usage violation and the correct pattern, ExampleCheck generates a short warning message that describes the violation in natural language. Table 1 shows the warning message templates for different types of API usage violations. In each template, <?> is instantiated with the corresponding API calls or control constructs based on the detected API usage violation and the correct pattern. <before/after> is instantiated based on the relative order of the two API calls in the correct pattern. Note that though *missing try-catch* is a subtype of *missing control construct*, we design a specialized template to elaborate the exception type for a missing-try-catch violation. To help users understand the prevalence of the suggested API usage pattern, the warning message also quantifies how many other code fragments follow the same pattern in GitHub.

Fix suggestion. ExampleCheck further suggests the correct way of using an API method by synthesizing a readable fixed snippet based on the context of the original SO snippet. ExampleCheck first matches each API call or control construct in the API usage pattern with the method call sequence of the SO snippet. If an API call or a control construct is matched with a code element in the original SO snippet, ExampleCheck directly copies the corresponding code element from the SO snippet to the synthesized snippet. Otherwise, ExampleCheck generates a method call statement and names the receiver and arguments based on their types. For example, if the receiver type of an unmatched API call (i.e., a *missing-API-call violation*) is `File`, ExampleCheck names the receiver variable as `file`, the lower case of the receiver type. By contextualizing the correct pattern based on the original SO snippet, ExampleCheck can reduce the mind gap when a programmer switches between the original post and the fixed snippet.

[4] https://www.w3schools.com/bootstrap/bootstrap_popover.asp

Violation Type	Description Template	Example Warning Message
Missing/Incorrect Order of API calls	You may want to call <?> <before/after> calling <?>	You may want to call `TypedArray.recycle()` after calling `TypedArray.getString()`. [35784171]
Missing Control Constructs	You may want to call the API method <?> in <?>	You may want to call `Cursor.close()` in a finally block. [31427468]
Missing Try-Catch	You may want to handle the potential <?> exception thrown by <?> using a try-catch block	You may want to handle the potential `SQLException` thrown by `PreparedStatement.setString()` using a try-catch block. [11183042]
Incorrect Guard Conditions	You may want to check whether <?> is true before calling <?>	You may want to check whether `iterator.hasNext()` is true. [25789601]

Table 1: Warning message templates for different types of API usage violations. <?> and <before/after> are instantiated based on API usage violations and correct patterns. The digits in the last column are the SO post ids of the warning examples.

4 RELATED WORK

Prior work has investigated the quality of code snippets in Stack Overflow in different perspectives. Several studies show that SO snippets are often incomplete and the API names appearing in these snippets are hard to resolve [3, 12, 15]. Zhou et al. observe that 86 of 200 accepted SO posts use deprecated APIs but only 3 of them are reported by other users [18]. Fischer et al. find that 29% of security-related SO snippets are insecure and have potentially been reused to over 1 million Android apps on Google play [6]. Treude and Robillard conduct a survey to investigate comprehension difficulty of code examples in Stack Overflow [13]. The responses from GitHub users indicate that less than half of the SO examples are self-explanatory due to issues such as incomplete code and missing explanations. Though we draw motivation from these studies, ExampleCheck focuses on detecting API usage violations by contrasting SO code examples against common API usage patterns mined from GitHub. While ExampleCheck follows a similar style to Codota [1], Codota does not group related examples based on common API usage, does not quantify how many GitHub code snippets support the common usage, and does not detect API misuse by contrasting the SO post against the usage.

5 SUMMARY

This paper introduces a Chrome extension, ExampleCheck that proactively detects API usage violations in a Stack Overflow post and enriches the post with extra API usage tips evidenced by a large number of GitHub code examples. Certainly, SO snippets are supposed to provide a starting point, not necessarily being fully complete or reliable. However, such incomplete or unreliable snippets can potentially impact the production code, when a programmer mentally or physically reuses a snippet to a target project. ExampleCheck can help programmers implicitly assess a given SO snippet and reduce the effort of cross-checking and testing the snippet when reusing it to a target project.

Future work. We plan to conduct a longitudinal study with Stack Overflow users to understand the usefulness of ExampleCheck. We have published ExampleCheck in Chrome Web Store and are in the process of intrumenting ExampleCheck to log user interaction behavior. By analyzing user behavior and conducting post surveys, we can gain both quantitative and qualitative insights of the adoption and usage of ExampleCheck as well as improvement opportunities. We also want to design code completion tasks and conduct a controlled lab study to evaluate whether using ExampleCheck indeed helps participants write more reliable code.

REFERENCES

[1] [n. d.]. Codota Code Browsing Assistant. https://www.codota.com/code-browsing-assistant. Accessed: 2017-11-13.

[2] Joel Brandt, Philip J Guo, Joel Lewenstein, Mira Dontcheva, and Scott R Klemmer. 2009. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1589–1598.

[3] Barthélémy Dagenais and Martin P Robillard. 2012. Recovering traceability links between an API and its learning resources. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 47–57.

[4] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[5] Ekwa Duala-Ekoko and Martin P Robillard. 2012. Asking and answering questions about unfamiliar APIs: An exploratory study. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 266–276.

[6] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. 2017. Stack overflow considered harmful? the impact of copy&paste on android application security. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 121–136.

[7] Natalie Gruska, Andrzej Wasylkowski, and Andreas Zeller. 2010. Learning from 6,000 projects: lightweight cross-project anomaly detection. In *Proceedings of the 19th international symposium on Software testing and analysis*. ACM, 119–130.

[8] Bin Liang, Pan Bian, Yan Zhang, Wenchang Shi, Wei You, and Yan Cai. 2016. AntMiner: mining more bugs by reducing noise interference. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 333–344.

[9] Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H. Pham, Jafar M. Al-Kofahi, and Tien N. Nguyen. 2009. Graph-based mining of multiple object usage patterns. In *ESEC/FSE '09: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, New York, NY, USA, 383–392. https://doi.org/10.1145/1595696.1595767

[10] Caitlin Sadowski, Kathryn T Stolee, and Sebastian Elbaum. 2015. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 191–201.

[11] Jamie Starke, Chris Luce, and Jonathan Sillito. 2009. Working with search results. In *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*. IEEE Computer Society, 53–56.

[12] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. 2014. Live API documentation. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 643–652.

[13] Christoph Treude and Martin P Robillard. 2017. Understanding Stack Overflow Code Fragments. In *Proceedings of the 33rd International Conference on Software Maintenance and Evolution*. IEEE.

[14] Jue Wang, Yingnong Dang, Hongyu Zhang, Kai Chen, Tao Xie, and Dongmei Zhang. 2013. Mining succinct and high-coverage API usage patterns from source code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 319–328.

[15] Di Yang, Aftab Hussain, and Cristina Videira Lopes. 2016. From query to usable code: an analysis of stack overflow code snippets. In *Proceedings of the 13th International Workshop on Mining Software Repositories*. ACM, 391–402.

[16] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. 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*. ACM, 886–896.

[17] Hao Zhong, Tao Xie, Lu Zhang, Jian Pei, and Hong Mei. 2009. MAPO: Mining and recommending API usage patterns. In *European Conference on Object-Oriented Programming*. Springer, 318–343.

[18] Jing Zhou and Robert J Walker. 2016. API deprecation: a retrospective analysis and detection method for code examples on the web. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 266–277.