Augmenting Stack Overflow with API Usage Patterns Mined from GitHub

Anastasia Reinhardt1,2∗ Tianyi Zhang1† Mihir Mathur1 Miryung Kim1
1University of California, Los Angeles
2George Fox University
areinhardt14@georgefox.edu, {tianyi.zhang, miryung}@cs.ucla.edu, mihirmathur@ucla.edu

ABSTRACT
Programmers often consult Q&A websites such as Stack Overflow (SO) 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 SO posts using API usage patterns mined from 380K GitHub projects. It quantifies how many GitHub examples follow common API usage and illustrates how to remedy the detected violation in a given SO snippet. With ExampleCheck, programmers can easily identify the pitfalls of a given SO snippet and learn how much it deviates from common API usage patterns found in GitHub. ExampleCheck is available at Chrome Web Store [2] and the demo video is available at https://youtu.be/WOnN-wQZsH0.

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:

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 [8]. Stack Overflow (SO) is a popular Q&A website that programmers often resort to. As of July 2017, Stack Overflow has accumulated more than 22 million answers, many of which contain code snippets 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 during software development. For example, Fischer et al. find that 29% of security-related code snippets in Stack Overflow are insecure and might affect over 1 million Android apps in Google play [7].

This tool demonstration paper builds on the API usage mining and API misuse detection technique described in our ICSE 2018 paper [11]. Our insight is that common API usage inferred from a large corpus of 380K GitHub projects may represent a desirable pattern that a programmer can use to examine and enhance SO code snippets. Mined API usage patterns abstract away syntactic details such as variable names, but retain the temporal ordering, control structures, and guard conditions of API calls.

This paper, in particular, focuses on the tool features and implementation details of a Chrome extension, called ExampleCheck that informs programmers about API usage violations in SO posts. Figure 1 shows a screenshot of ExampleCheck. 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 against the common API usage patterns mined from GitHub. To help users understand a detected violation, ExampleCheck generates a descriptive warning message and illustrates how to fix the violation with corresponding GitHub examples. Using common API usage patterns to detect violations may lead to false alarms, since these mined, common patterns do
2 APPROACH & IMPLEMENTATION

This section describes the tool implementation details of **ExampleCheck**. Figure 2 shows the architecture of **ExampleCheck**. The API usage mining process is computed offline and the resulting patterns are stored in a database. The technical details and evaluation of API usage mining technique is presented in our ICSE 2018 paper [11]. 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 warning messages and corresponding fixes. For each misused method call in a snippet, the Chrome extension generates a pop-up window using the Bootstrap popover plug-in to inform the user about the API misuse information.

**API Usage Mining and the Resulting Pattern Set.** Our mining technique in [11] leverages a distributed software mining infrastructure [6] to search over the corpus of 380K GitHub projects. Given an API method of interest, it identifies code fragments that use the same method in the GitHub corpus and performs program slicing to remove statements that are not related to the given method. Then it combines frequent subsequence mining and SMT-based guard condition mining to retain important API usage features, including the temporal ordering of related API calls, enclosing control structures, and guard conditions that protect an API call. We evaluated the mining technique using 30 API methods from MUBench [3]. Our mining technique has 80% precision and 91% recall, when considering top 5 patterns for each API method.

In our prior work [11], we mined API usage patterns of 100 popular Java API methods and carefully inspected 245 inferred patterns based on online documentation. As a result, we curated a dataset of 180 validated, correct patterns for API misuse detection, which covers API usages shown in 217K SO posts in Java. 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.

**API Misuse Detection.** Given a code snippet sent from the browser, the server first extracts the API call sequence from the snippet. We use a partial program analysis and type resolution technique [9] to parse incomplete snippets and resolve ambiguous types. If a SO snippet has 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** 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 an API call in the sequence implies the guard of the corresponding API call in the pattern. **ExampleCheck** uses a SMT solver, Z3 [5], to check whether one guard condition implies another. **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 warning message that describes the violation in natural language text. 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. The warning messages also describe which exception types are not handled in the snippets detected with *missing try-catch* violations. To help users understand the prevalence of a recommended API usage pattern, the warning message also quantifies how many other code fragments follow the same pattern in GitHub.

**Fix Suggestion.** **ExampleCheck** further suggests a correct way of using an API method by synthesizing a readable fixed snippet based on the original SO snippet. **ExampleCheck** first matches each API call in the recommended API usage pattern with the given SO snippet. If an API call is matched, **ExampleCheck** reuses the same receiver object and arguments of the corresponding API call from the original SO snippet in the synthesized snippet. Otherwise, **ExampleCheck** 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 object as `f11e`, the lower case of the receiver type. In this

---

1. [https://www.w3schools.com/bootstrap/bootstrap_popover.asp](https://www.w3schools.com/bootstrap/bootstrap_popover.asp)
way, ExampleCheck reduces the mental gap for switching between the original SO post and the recommended snippet.

### 3 Demonstrations Scenario

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, as shown in Figure 1. Though this post is accepted as a correct answer, it does not properly use the JsonElement.getAsString method, 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 will return null, which consequently leads to NullPointerException when calling getAsString on the returned object. If Alice puts too much trust on this example of the SO post, she may inadvertently follow an unreliable solution, which might lead to runtime errors in some corner cases.

Alice cannot easily recognize the potential limitation of the given SO post, unless she manually investigates other similar code examples. ExampleCheck frees Alice from this manual investigation labor by contrasting a Stack Overflow post with common API usage patterns mined from over 380K GitHub repositories. ExampleCheck then highlights the 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 natural language description (① in Figure 1). From the warning message, Alice learns that she should check whether the JsonElement object is null before calling getAsString. Furthermore, ExampleCheck displays a message that 119 GitHub examples also follow this API usage pattern. Such quantification can provide additional evidence about how many real-world examples are different from the given SO snippet.

**Fix suggestion.** ExampleCheck further sketches how to correct 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 reduce the gap between the fix and the original post, ExampleCheck reuses the same variable names in the original SO posts to generate a suggestion with improved API usage. For example, the JsonElement variable in the generated example is named as the same variable, match_number in the original post.

**Linking GitHub examples.** To help Alice understand how the same API method is used in real-world projects, ExampleCheck provides several GitHub examples that follow the suggested API usage pattern ③ in Figure 1. Alice is curious about how others use JsonElement.getAsString. When 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 3. 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 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 a 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 (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.

---

[2] https://goolFYHo1UM
Multiple API usage violations. If a method call in a SO post violates multiple API usage patterns, ExampleCheck displays them in separate pages in a pop-up window. These pages are first ranked by the vote score of each violated pattern, e.g., upvotes minus downvotes. If two patterns have the same vote score, ExampleCheck then ranks them by the number of GitHub examples that support a pattern. As shown in ⑥ in Figure 1, the method call, `getAsString` violates four API usage patterns. Figure 4 shows the second violated pattern and suggests Alice to check whether the `JsonElement` object represents a JSON primitive value before calling the `getAsString` method. 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.

4 RELATED WORK

Prior work has investigated the quality of code snippets in Stack Overflow. Several studies show that SO snippets are often incomplete and the API names appearing in these snippets are hard to resolve [4, 9]. Zhou et al. observe that 86 of 200 accepted SO posts use deprecated APIs but only 3 of them are reported by other users [12]. 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 [7]. Treude and Robillard conduct a survey to investigate comprehension difficulty of code examples in Stack Overflow [10]. 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 desirable API usage mined from GitHub.

REFERENCES


5 SUMMARY

The main contribution of this paper is the design and implementation of ExampleCheck, which provides browser-based tool support for systematically assessing and augmenting Stack Overflow with common API usage patterns mined from GitHub. In our previous work [1], we examine 217K SO posts with 180 validated patterns and find that 31% of SO posts have potential API usage violations. The first two authors manually inspect 400 SO posts with detected API usage violations and confirm real API misuse in 289 posts, which can produce symptoms such as program crashes and resource leaks if the posts are reused as-is to target projects. We also find that many unreliable examples are simplified to operate on crafted input data for illustration purposes only. Such curated examples could be insufficient for various input data and usage scenarios in real software systems, especially for handling corner cases.

As future work, we plan to conduct a longitudinal study with Stack Overflow users to understand the adoption and usage of ExampleCheck. We will also investigate different weighting schemes for effectively ranking detected API usage violations.