Interactively Augmenting Stack Overflow with API Usage Patterns Mined from GitHub

Anastasia Reinhardt† Tianyi Zhang* Miryung Kim*
†George Fox University, Newberg, OR, USA *University of California, Los Angeles, USA

ABSTRACT

Programmers often consult Q&A websites like Stack Overflow to learn new APIs. However, such online code examples are not always complete or reliable in terms of API usage. To assess and augment Stack Overflow examples, we present an interactive approach, Soap, that contrasts code examples with API usage patterns learned from GitHub and alerts users about the potential API usage violations. With the assistance of Soap, programmers do not need to cross-check multiple examples for proper API usage reference, and can also build confidence by learning how many other GitHub developers also follow the same practice.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments—Integrated Environments

1. INTRODUCTION

This paper's main contribution is to describe the features of Soap from a user's perspective. This user interface, in the form of a Chrome extension, was developed to make a previous project, Maple [9], accessible for programmers.

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 [6]. Stack Overflow (SO) is a popular Q&A website that programmers often consult. As of July 2017, Stack Overflow has accumulated more than 22 million answers, many of which contain code examples to demonstrate the solution for a particular programming question. However, SO examples are not always complete or reliable, which can be misleading and potentially dangerous when programmers follow the same example to complete a client program. Our previous study shows that over 50% of 31,801 SO posts contain API misuse that could produce symptoms of program crashes and resource leaks if reused in a target system [9]. Neglecting to close an input stream, for example, could lead to data leakage or generally unreliable code.

To assess online code examples, this paper presents Soap, an interactive approach that augments Stack Overflow with code idioms learned from GitHub and alerts programmers about potential violations in a code example. Soap leverages a scalable API usage mining technique from Maple [9] to learn three types of API-related idioms—temporal ordering, guard conditions, and exception handling of API calls—from over 7 million GitHub projects. Our insight is that commonly practiced idioms in massive code corpora may represent a desirable pattern that a programmer can use to trust and enhance code examples on Stack Overflow.

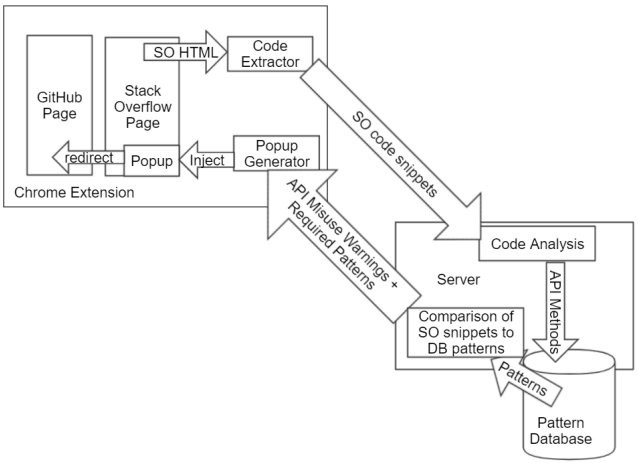


Figure 1: An overview of Soap's architecture

Given an SO example, Soap first extracts the sequence of API calls with corresponding control constructs and guard conditions. Soap then contrasts the sequence with commonly practiced idioms in GitHub and highlights code regions that violate the idioms. To help users better understand the violations, Soap further generates descriptive warning messages and also contextualizes a violated idiom by synthesizing a fixed example. Mining code idioms to detect API usage violations often suffers from reporting false alarms, since mined idioms may not be inclusive and fit all usage scenarios of an API [3]. To mitigate this issue, Soap allows users to upvote or downvote a violation based on its applicability and usefulness to an SO example. Soap filters a code idiom when multiple users flag it as unhelpful to assess an example. To help developers build confidence on a code idiom, Soap shows how many GitHub developers also follow the idiom as well as how many other users like or dislike this idiom.

A user of Soap would benefit from the addition of examples from compiled GitHub resources to the code examples she encounters on Stack Overflow. This will not only combat programming issues stemming from the use of incomplete or unreliable SO code examples, 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 she will learn common and reliable usage patterns for a given API.

In order to give programmers access to Soap, the front-end of Soap is implemented as a Chrome extension that users can easily download and install.

2. MOTIVATING EXAMPLES AND TOOL FEATURES

Consider Alice, a software developer who needs to convert Java objects to their JSON representations using Google's Gson library[2], which she is unfamiliar with. Alice finds a Stack Overflow post that demonstrates how to get the string value of a json element, as shown in Figure 2. However, this example does not use the `JsonElement` API properly.

Detecting and Highlighting Potential API Misuse. The pattern mining technique learns two `JsonElement.getAsString` patterns that are commonly practiced by GitHub developers: (1) a check to make sure that the `JsonElement` object is of the type `JsonPrimitive` by using `JsonElement.isJsonPrimitive` before calling `getAsString`, and (2) a check to make sure that the `JsonElement` object is not null before calling `getAsString`.

The extension highlights the potential API misuse in the code snippet, as seen in Figure 2. Alice is interested in learning more about the API and what specifically the code snippet did not include, so she clicks on the highlighted text.

Stack Overflow Popup View. Clicking on the highlighted text reveals a popup, as seen in Figure 2a and 2b. The popup is populated with information about any required patterns in Soap's database this particular API call does not adhere to. Alice notices that there are two pages of the popup, indicating two different usage patterns that this call does not follow, as shown in Figure 3a and Figure 3b.

Alice inspects the first page (Figure 3a) and sees a warning message, automatically generated by Soap for that particular API misuse. She learns that she should check whether the `JsonElement` object is null before calling `getAsString`. She notices that 113 other GitHub code examples use this pattern, which gives her a quantitative measurement of how prevalent this pattern is in real-world projects. Below this is a code example following the required pattern, generated by Soap based on the context of the SO example.

Alice then inspects the second page (Figure 3b), and finds that it suggests to check whether the `JsonElement` object is primitive before calling `getAsString` to avoid a `ClassCastException`. She notices that this pattern has less than half the support of the previous pattern.

Curious to see the first pattern in context, Alice returns to the first page of the popup and clicks on the first link provided to her under "See this in a GitHub example."

GitHub Example View. When Alice clicks on one of the GitHub links, the file opens in a new tab and the view scrolls to where the API is called in the file, and the method in which this occurs is highlighted so Alice can easily find it, as seen in Figure 4. The addition of a compilable code example that demonstrates the pattern in context can aid Alice in understanding how to use the pattern if it is unfamiliar to her. In this case, Alice finds herself redirected to the method in a GitHub project seen in Figure 4.

Returning to the popup in Stack Overflow, Alice clicks on the second link provided for the second page to compare usage patterns in context. This link opens up to the GitHub method seen in Figure 4b. She notices that the example uses a null check in conjunction with the primitive check, which makes sense to her after seeing that both were missing from the Stack Overflow code snippet. However, she notes that the more commonly-used pattern only uses a null check.

After seeing these two examples, Alice can infer that a null check is more necessary and is more common than the primitive check, based on the GitHub examples she has seen as well as the GitHub support indicated by the popup message. She upvotes the null check's pattern by clicking on the up-arrow on its page (see Figure 3b) to send the server her feedback on the patterns it gave her.

3. IMPLEMENTATION

This section describes the implementation details of Soap. The front-end is implemented as a Chrome extension and the back-end is set up in an Apache server. Soap consists of five components: (1) Stack Overflow code snippet extraction, (2) API misuse detection, (3) popup generation, (4) GitHub example alteration, and (5) a user feedback interface.

Code Snippet Extraction. When a user loads a Stack Overflow webpage, the Chrome extension extracts code snippets within `<code>` tags in answer posts, and sends them in a JSON message to the server, as seen in Figure 1.

API Misuse Detection. When the server receives a message from the plug-in, it parses the snippets into API call sequences, abstracting away irrelevant statements and syntactic details. An API call sequence consists of relevant control constructs and API calls, where the API calls are annotated with the number of arguments as well as any guard conditions associated with it.

Pattern Mining. Once the code snippets are parsed, Soap searches its pattern database for the API calls present in each API call sequence. Soap's pattern database is bootstrapped with mined patterns of 30 well-studied Java and Android APIs. When Soap encounters a new API that does not exist in its database, it automatically issues another mining request and expands the database with the new patterns it finds. Specifically, Soap utilizes a distributed software mining infrastructure, Boa,[5] to traverse the abstract syntax trees (ASTs) of 7 million Java projects, collected September 2015 from GitHub. For every AST method, Soap checks if it is from the API of interest. If it is, Soap translates the code snippet into a structured call sequence. From these call sequences, Soap finds a common subsequence which is the required pattern for that API method, and this is added to the database.

When Soap receives the required and alternative patterns associated with the specified API calls, it checks whether the code snippet's call sequence satisfies one of the alternative patterns and all required patterns. A code snippet's call sequence satisfies a pattern if it is subsumed by it.

Additionally, while this checking process is occurring, the guard conditions in the code snippets are generalized before checking their logical implications using the SMT solver Z3. Z3 is used to check the logical equivalence between two guard conditions so that Soap can merge logically-similar clusters of guard conditions.

[1]https://stackoverflow.com/questions/34120882/gson-jsonelement-getasstring-vs-jsonelement-tostring
[2]https://github.com/google/gson/blob/master/UserGuide.md
[4]http://tinyurl.com/JsonObjectDriver
[4]http://tinyurl.com/SpoutcraftConfig
[5]http://boa.cs.iastate.edu/

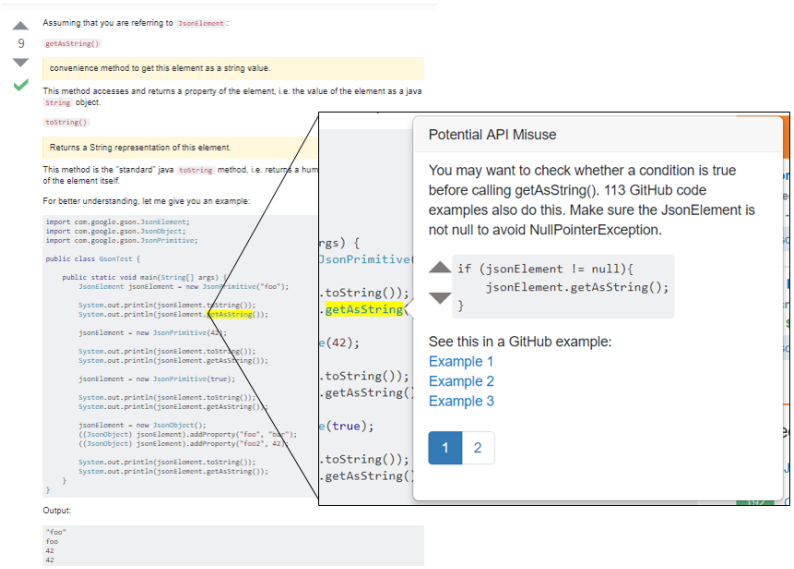
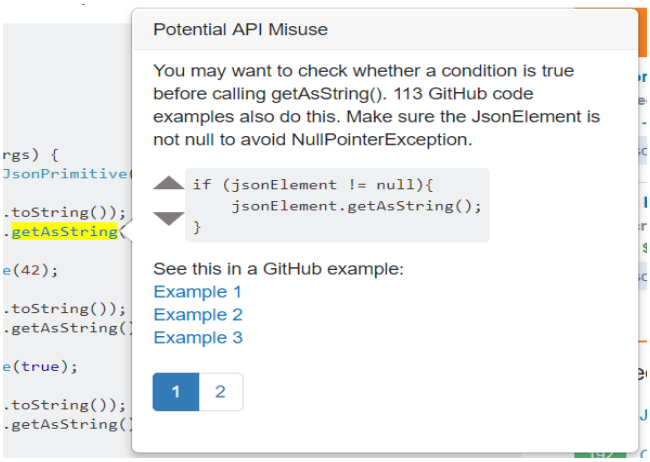
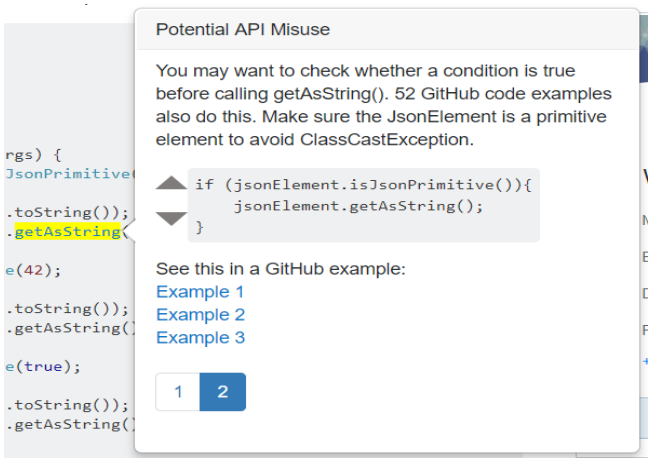


Figure 2: A code snippet that does not properly check `JsonElement.getAsString`.[1]



(a) A page describing a way to avoid a `NullPointerException` by checking whether the `JsonElement` object is null.



(b) A page describing a way to avoid a `ClassCastException` by checking whether the `JsonElement` object is a primitive.

Figure 3: The two pages of a popup generated on `JsonElement.getAsString`.

If SOAP finds that an API call sequence does not satisfy the necessary patterns, it generates violations for each potential API misuse and returns them to the plug-in.

Popup Generation. Using the data from the server's JSON message, the plug-in searches the identified code snippet for the API call in question, highlights it, and generates a Bootstrap popover on it, as seen in Figure 3. The popover is populated with a violation message describing the pattern being violated and including the GitHub support for the required pattern, in terms of number of supporting projects as well as three links to relevant GitHub pages using that pattern correctly. The plug-in generates one popover for each API call, and generates pages of the popover for each different pattern being violated by that call. The provided example of the pattern is currently hard-coded.

GitHub Example Alteration. When the user clicks on a provided GitHub example in the popup, the plug-in's main script writes the name of the method call associated with that link to a shared storage space for the Chrome extension, using the `chrome.storage` API. The script highlights the entire method and scrolls the view down so the user is easily able to find it, as seen in Figure 4.

User Feedback. Users are able to give feedback on the patterns the popup shows them by voting "up" or "down" on them. When a vote is registered, the plug-in sends the server a message with the pattern's specific ID and a vote of +1 or -1, which the server uses to update the database. This will be used in the future to rank patterns that are sent to the plug-in, and learn which ones users find helpful or unhelpful.

4. RELATED WORK

Quality Assessment of Online Code Examples. The quality of online code examples has been investigated from various perspectives, with the conclusion that such examples

```
166     @Override
167     public void decimalProperty(PropertyName name, JsonObject context) throws IOException {
168         JsonElement prop = property(context, name);
169         if (prop == null) {
170             return;
171         }
172         builder.add(name, prop.getAsBigDecimal());
173     }
174
175     @Override
176     public void stringProperty(PropertyName name, JsonObject context) throws IOException {
177         JsonElement prop = property(context, name);
178         if (prop == null) {
179             return;
180         }
181         builder.add(name, prop.getAsString());
182     }
183
184     private static final Pattern DATE = Pattern.compile("(\\d{3,})-(\\d{1,2})-(\\d{1,2})");
185     @Override
186     public void dateProperty(PropertyName name, JsonObject context) throws IOException {
187         JsonElement prop = property(context, name);
188         if (prop == null) {
189             return;
190         }
```

(a) The second GitHub example for Figure 3a in the context of its GitHub file.[3]

```
99
100         private static String getSelectedLauncherProfile(File minecraftDir) {
101             File path = new File(minecraftDir, "launcher_profiles.json");
102             JsonObject json = JsonUtils.parseJson(path);
103
104             if (json != null) {
105                 JsonElement element = json.get("selectedProfile");
106
107                 if (element != null && element.isJsonPrimitive()) {
108                     return element.getAsString();
109                 }
110             }
111
112             return null;
113         }
114
```

(b) The first GitHub example for Figure 3b.[4]

Figure 4: The GitHub examples redirected to from the links provided in the popup, highlighted by the Chrome extension.

are largely unreliable in terms of resolvable, parsable, or compilable code. Dagenais and Robillard [1] find that 89% of API names in online-forum code snippets are ambiguous and not easily resolved due to their incompleteness. Subramanian et al. [7] observe that 66% of a corpus of 39,000 SO snippets are free-standing statements, lacking class or method declarations. Similarly, Yang et al. [8] find that out of 91,000 Java SO code snippets, only 3.89% are parsable and 1% are compilable. Zhou et al. [10] observe that 86 of 200 accepted SO posts use deprecated APIs but only 3 of them are reported by other users. We previously performed a study [9] considering the implications of following such code examples verbatim and how this may encourage potential API misuse and lead to unexpected behavior in client programs.

API Usage Mining. Soap's API usage mining is described in further detail in our previous technical study. A large body of literature in API usage mining currently exists which uses a variety of techniques. However, to our knowledge these existing pattern mining techniques do not mine from massive code corpora with millions of projects. Gruska et al. [2] mines from the largest code corpus we are aware of, which comprises 6,000 Linux projects. Our pattern mining also uses a prediate mining technique to mine API call guard conditions in addition to API call ordering. To our knowledge, Ramanathan et al. [5] and Nguyen et al. [4] are the only other predicate mining techniques, and unlike these techniques we formalize the predicate equivalence problem as a satisfiability problem and leverage an SMT solver to group logically equivalent predicates during guard mining [9].

Codota. The Codota Code Browsing Assistant for Chrome[6] analyzes Java and Android code snippets in web pages and enhances them with IDE-like features. A user may click on underlined code to view API documentation, references, and API-level compatibility issues, and may also save code snippets to view later in a CodeBox supported by Codota. Soap, which draws inspiration from this tool's user interface, instead focuses specifically on potential API misuse and does so by inferring common usage patterns from GitHub code projects, as opposed to using API documentation. Soap also uses code examples derived from required patterns as well as code examples in their original GitHub context to demonstrate a given pattern to a user as opposed to describing it with natural language or API documentation.

[6]https://www.codota.com/code-browsing-assistant

5. SUMMARY

Soap is an interactive approach to enriching code examples on Q&A sites like Stack Overflow that incorporates API usage mining and user feedback in its UI design. Programmers often use sites like Stack Overflow to understand API usage, but the reliability of the code examples on these forums is under question. This user interface was designed with the intention of reducing the need to cross-check sites for API usage questions. It takes advantage of the benefits of using easy-to-understand code examples as opposed to API documentation for learning API usage by enhancing the examples already present in Stack Overflow while also providing other, more reliable ones.

For future work, we would like to qualify the benefits of Soap with a user study, as well as generate code examples for the popups based on the abstract patterns we have in the database.

6. REFERENCES

[1] B. Dagenais and M. P. Robillard. Recovering traceability links between an api and its learning resources. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 47–57. IEEE, 2012.

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

[3] B. Liang, P. Bian, Y. Zhang, W. Shi, W. You, and Y. Cai. Antminer: mining more bugs by reducing noise interference. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 333–344. IEEE, 2016.

[4] H. A. Nguyen, R. Dyer, T. N. Nguyen, and H. Rajan. Mining preconditions of APIs in large-scale code corpus. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 166–177. ACM, 2014.

[5] M. K. Ramanathan, A. Grama, and S. Jagannathan. Static specification inference using predicate mining. In *ACM SIGPLAN Notices*, volume 42, pages 123–134. ACM, 2007.

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

[7] S. Subramanian and R. Holmes. Making sense of online code snippets. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 85–88. IEEE Press, 2013.

[8] D. Yang, A. Hussain, and C. V. Lopes. From query to usable code: an analysis of Stack Overflow code snippets. In *Proceedings of the 13th International Workshop on Mining Software Repositories*, pages 391–402. ACM, 2016.

[9] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim. Are code examples on an online Q&A forum reliable? A study of API Misuse on Stack Overflow. In Submission for ICSE, 2017.

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