API Caveat Explorer: Surfacing Negative Usages from Practice

An API-oriented Interactive Exploratory Search System for Programmers

Jing Li*, Aixin Sun*, Zhenchang Xing†, and Lei Han*
*School of Computer Science and Engineering, Nanyang Technological University, Singapore
†College of Engineering and Computer Science, Australian National University, Australia
jli030@e.ntu.edu.sg;axsun@ntu.edu.sg;zhenchang.xing@anu.edu.au;tomhanlei@hotmail.com

ABSTRACT

Application programming interface (API) documentation well describes an API and how to use it. However, official documentation does not describe "*how not to use it*" or the different kinds of errors when an API is used wrongly. Programming caveats are *negative usages* of an API. When these caveats are overlooked, errors may emerge, leading to heavy discussions on Q&A websites like Stack Overflow. In this demonstration, we present API Caveat Explorer, a search system to explore API caveats that are mined from large-scale unstructured discussions on Stack Overflow. API Caveat Explorer takes API-oriented queries such as "HashMap" and retrieves API caveats by text summarization techniques. API caveats are represented by sentences, which are *context-independent*, *prominent*, *semantically diverse* and *non-redundant*. The system provides a web-based interface that allows users to interactively explore the full picture of all discovered caveats of an API, and the details of each. The potential users of API Caveat Explorer are programmers and educators for learning and teaching APIs.

ACM Reference Format:
Jing Li*, Aixin Sun*, Zhenchang Xing†, and Lei Han*. 2018. API Caveat Explorer: Surfacing Negative Usages from Practice: An API-oriented Interactive Exploratory Search System for Programmers. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3209978.3210170

1 INTRODUCTION

Application programming interfaces (APIs) are foundations of software development [3, 6, 9]. To program to an API, programmers need to know not only "how to use an API", but also "how **not** to use the API". Table 1 lists three example negative usages of APIs extracted from Stack Overflow[1], a popular Q&A website for topics in programming. In this demonstration, we refer to such *negative usages* as **API caveats**.

Especially, API documentation is an important resource for programmers to learn unfamiliar APIs. By providing important information about functionality, parameters and usage scenarios of an API, official API documentation often does a good job at explaining "how to use an API" [9]. However, API documentation does not mention API caveats as it is hard to predict the individual practical use cases of APIs. For example, Java class JTextArea (the second example in Table 1) is designed to display plain text only; as such, it does not support styled text. Unfamiliar with this class, a programmer may wonder why it fails to display styled text. An effective way of seeking solution is to post a question on Stack Overflow, and wait for suggestions from other programmers. Often, the answers explicitly point out the overlooked API caveat, as shown in Table 1.

Table 1: Examples of API caveats.

API Types	API caveats extracted from Stack Overflow
HashMap	"**Don't** use a HashMap if you are going to have multiple threads, use a ConcurrentHashMap instead."
JTextArea	"JTextArea **is not** a component designed for styled text."
ActionListener	"Inner classes, such as your ActionListener, **cannot** access non-final variables from the scope that contains it."

Alternative to official documentation, the Q&A discussions in fact "document" the API caveats emerging from practical use cases. On the other hand, the crowd-generated Q&A discussions are informal, redundant, and sometimes related to very specific use cases. To discover API caveats from massive noisy discussions, and to present the discovered API caveats in an organized and concise form, are both challenging tasks. In our research, we formulate the task of discovering API caveats as a text summarization task, which is to find representative sentences for each API caveat.

To the best of our knowledge, API Caveat Explorer is the first system to tackle the problem of negative usages of APIs. API Caveat Explorer supports three main functionalities. *First*, it automatically surfaces API caveats from large-scale unstructured Q&A discussions using text summarization techniques. *Second*, it ensures the sentences representing the discovered API caveats are context-independent, prominent, semantically diverse, and non-redundant. While diversity and redundancy are typical requirements in text summarization tasks, context-independency and prominence are essential in our task because the discovered API caveats have to be "common" negative usages, not very specific to some unique cases, and also be representative. *Third*, the system provides an API-oriented search system, which allows users to have a bird's-eye

[1] https://stackoverflow.com/

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5657-2/18/07.
https://doi.org/10.1145/3209978.3210170

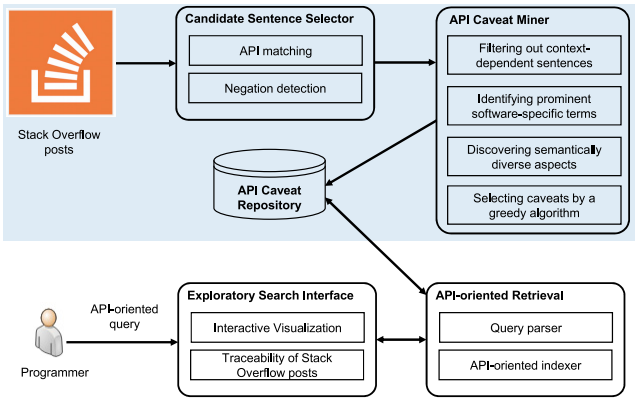

Figure 1: The system architecture of API Caveat Explorer. The upper half shows the process of mining API caveats and the lower half shows the search function.

view of all discovered caveats of an API, as well as to zoom into details of a specific caveat.

2 API CAVEAT EXPLORER

Figure 1 illustrates the system architecture of API Caveat Explorer. We will detail the four major modules in Section 2.1. After that, we brief evaluation of the system in Section 2.2.

2.1 System Architecture

Given a collection of posts (*i.e.,* questions and answers) from Stack Overflow, the two modules shown in the upper half of Figure 1 mine all API caveats, and store them in a repository. Specifically, *Candidate Sentence Selector* selects the sentences that mention APIs and with negative expressions, as candidate sentences. The *API Caveat Miner* is responsible for discovering desirable API caveats from the candidate sentences. These processes are offline. The two modules shown in lower half of Figure 1 support the searching function. The *Exploratory Search Interface* provides interactive exploration of API caveats, that are retrieved by *API-oriented Retrieval* module in real time.

Candidate Sentence Selector. In this demonstration, we focus on classes and interfaces defined in Java SDK as the API types of interest. The raw sentences are extracted for Stack Overflow that are tagged with Java. We adopt a name-matching strategy to select sentences that mention a given API. More specifically, we develop a software-specific tokenizer to tokenize the sentences. This tokenizer preserves the integrity of code-like tokens, *e.g.,* java.util.HashMap. If a token in a sentence matches the full or partial name of an API, the sentence is considered mentioning the API. When selecting candidate sentences, variations of API mentions have to be taken into account. For example, mentions of "HashMap" include "hash map", "hashmaps" and "hash-map", in addition to the one defined in Java documentation, *i.e.,* HashMap.

As we are looking for negative usages of API, API caveats should be expressed in sentences containing negative expressions. To this end, we use a dependency parser to detect negative sentences. Dependency parse tree is a directed graph, where nodes represent words and edges represent dependency relations, *e.g., nsubj*: nominal subject, *aux*: auxiliary, *det*: determiner, etc.[2] We use negation modifier (*i.e., neg*) to detect negative expressions. To ensure the negative expressions are on APIs, we select only negative sentences whose subject or object is an API of interest.

After API matching and detection of negative expressions, we obtain a set of candidate sentences from Stack Overflow posts.

API Caveat Miner. Some of the candidate sentences are only meaningful with its context, for example, "HashMap cannot be used here". We therefore filter out context-dependent sentences, as we do not expect readers have to dig out the context to understand a sentence (*i.e.,* a caveat returned by our system). We remove context-dependent sentences from the candidate sentences, based on a set of sentence patterns, which are defined from observations made on the Stack Overflow sentences. The remaining sentences, *i.e.,* the context-independent sentences for API type x, are referred to as *candidate caveats*, denoted by C_x.

The second step is to identify prominent software-specific terms. An API caveat is usually concerned about software-specific terms related to the particular API usage, *e.g., thread-safe* and *sort*. Identifying prominent terms in C_x helps to distill frequently-overlooked but important API usage issues. For a term t in a sentence, we use relative entropy to weight its prominence: $w(t) = p(t) \log \frac{p(t)}{q(t)}$, where $p(t)$ is the probability of observing t in C_x and $q(t)$ is probability of observing t in all Stack Overflow posts that are tagged with Java.

Some of the prominent terms could be related closely *e.g., multi-thread* and *thread-safe*. The third step in API caveat miner is to cluster semantically-related prominent terms to discover semantic aspects of an API. This will in turn help to discover semantically diverse caveats. Here, semantic relatedness between terms is measured by term co-occurrence in sentences. Thus, we construct a term co-occurrence graph for C_x and use Louvain method [2] to cluster the term graph.

The last step of API caveat miner is to select sentences to represent each semantic aspect discovered in the earlier step. A semantic aspect is represented by a cluster of terms (*i.e.,* a term community). We formulate the selection of desirable caveats as a weighted *set cover* problem. Given an API type x, let T_x be a set of N prominent terms in one of its semantic aspects. The goal is to find a set cover $\mathcal{A}_x \subseteq C_x$ of minimal total cost to cover all terms in T_x, *i.e.,*

$$Minimize \sum_{s_i \in \mathcal{A}_x} cost(s_i), \text{ subject to } \bigcup_{s_i \in \mathcal{A}_x} s_i = T_x \qquad (1)$$

where the $costs(s_i)$ is computed from two parts: (i) average value of the prominence scores of terms in sentence s_i by $w(t)$, denoted by $prom(s_i)$, and (ii) post score $post(s_i)$. The post score is computed from the user votes of the post that contains sentence s_i. These two scores are normalized independently based on their corresponding maximum and minimal values. Then $costs(s_i)$ is a linear combination of the two scores, *i.e.,* $cost(s_i) = -\alpha \cdot prom(s_i) - \beta \cdot post(s_i)$ where α and β are the coefficients and $\alpha + \beta = 1$.

After sentence selection, for each API, we obtain a few semantic aspects. Each semantic aspect is represented by a set of prominent terms, and also a set of sentences. Overall all the sentences selected

[2] http://universaldependencies.org/en/dep/all.html

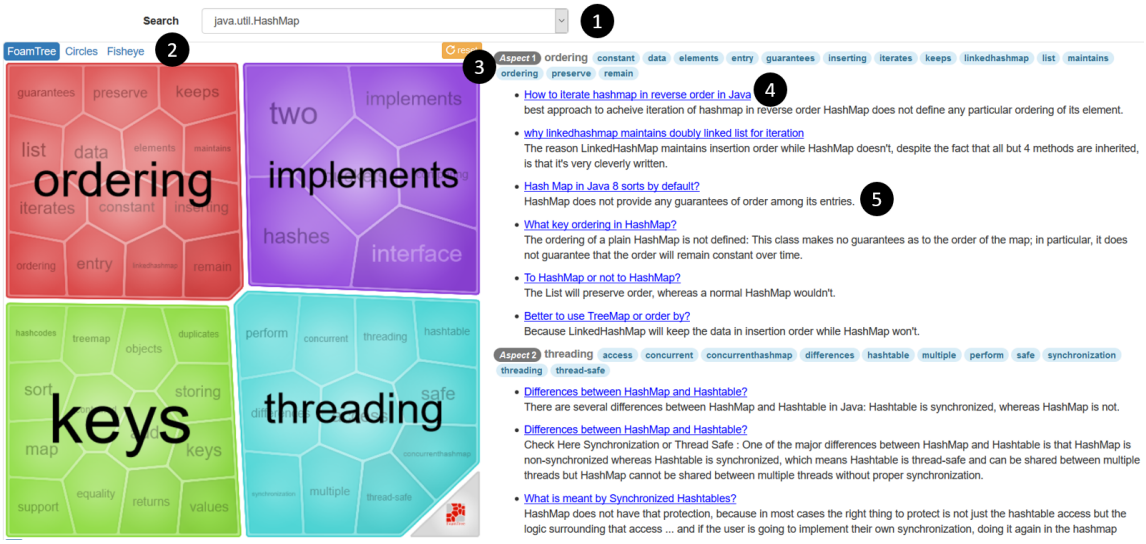

Figure 2: API Caveat Explorer screenshots (best viewed in color). The main GUI consists of five components: ① The query input panel. ② Interactive modes: FoamTree, Circles, and Fisheye. ③ Term community, representing a semantic aspect of an API. ④ Link to the original post on Stack Overflow. ⑤ The sentence detailing the API caveat.

for a given API ensure semantic diversity at aspect level, and ensure non-redundancy within each semantic aspect.

API-oriented Retrieval. We index the discovered caveats which mention the same API type as a virtual document. Given a query issued by a programmer, this module firstly parses the query to detect the full or partial API name. This module then retrieves all caveats for the API in query.

Exploratory Search Interface. To facilitate exploration of the discovered API caveats, we implemented a web interface with three interactive modes, using d3js[3] toolkit and FoamTree[4] framework. The main GUI consists of five components, as indicated on Figure 2.

(1) *The query input panel* is implemented with an API-aware auto-complete method, and the matched queries are displayed as user-friendly responsive drop-down menu.
(2) *The interactive panel* provides three modes to explore the discovered API caveats: *FoamTree* mode, *Circles* mode and *Fisheye* model, to be detailed shortly.
(3) *The term community* shows all prominent terms in a cluster. Each community represents one key semantic aspect of the API type.
(4) *The link* to the original post on Stack Overflow, from which this caveat is extracted, for accessing the discussion thread.
(5) *The caveat* shows the sentence (*i.e.,* API caveat) discovered by API Caveat Explorer, which mentions the queried API and contains negative expressions.

API Caveat Explorer implements three interactive modes, captured in Figure 3. *FoamTree* mode provides an engaging user experience with animated transitions and zooming, as an effective way to explore the details of API caveats. Unlike FoamTree mode which needs zooming, *Circles* mode provides the whole picture of discovered API caveats within a highly-interactive multi-level pie chart. *Fisheye* mode uses an interactive graph to show the relationships among the terms of API caveats. The node size is proportional to the degree centrality of the node in the graph. Different colors indicate different term communities. In short, the three interactive modes together facilitate users to explore and understand the discovered API caveats.

[3]https://d3js.org/
[4]https://carrotsearch.com/foamtree/

Table 2: Performance Comparison. †indicates that the improvements is statistically significant under paired t-test with $p \leq 0.05$.

Method	ROUGE-1	ROUGE-2	ROUGE-SU4
LDA	0.5473	0.3202	0.3550
KM	0.6026	0.3498	0.3836
LexRank	0.6045	0.3555	0.3923
MMR	0.6097	0.3583	0.3868
API Caveat Explorer	**0.6269†**	**0.4152†**	**0.4374†**

2.2 Effectiveness Analysis

Given an API type, API Caveat Explorer can extract a set of sentences as desirable caveats from the massive posts. Thus, we compare API Caveat Explorer with four classical text-summarization methods, each of which can independently select a subset of sentences as summaries (*i.e.,* caveats). Ten API types are used for evaluation in our experiments, with manual annotations as groundtruth. Specifically, we recruit three annotators who all have more than 4 years of programming experiences in Java to generate the gold standard summaries. We use three evaluation metrics, namely, ROUGE-1, ROUGE-2, ROUGE-SU4 for effectiveness analysis [7].

Figure 3: Exploration with three interactive modes: FoamTree mode, Circles mode, and Fisheye mode. In FoamTree and Circle mode, "cocurrent" is clicked.

Table 2 reports evaluation results of the five methods. API Caveat Explorer achieves the best performance against four baselines on ROUGE scores. More specifically, API Caveat Explorer achieves 22.47%, 12.25%, 10.66% and 10.60% improvements over Latent Dirichlet Allocation (LDA) [1], k-means (KM) [8], LexRank [4], and Maximal Marginal Relevance (MMR) [5], respectively. The improvements are statistically significant for the three kinds of ROUGE scores under paired t-test with $p \leq 0.05$. We attribute this to the fact that API Caveat Explorer takes *context independence*, *prominence*, *semantic diversity* and *non-redundancy* into account simultaneously when selecting API caveats.

3 DEMONSTRATION

Data and Settings. API Caveat Explorer relies on Java 8.0[5] APIs and Stack Overflow posts. From the official Java 8.0 website, we obtain 4,240 Java API types. We collect all Stack Overflow posts tagged with Java from the March-2016 data dump as the general corpus. Among the posts, 1,081,439 sentences mention at least one Java API type.

The configuration of API Caveat Explorer is based on the performance of a development set. Accordingly, for each API type, we use the top-100 prominent terms in its candidate sentences to construct the term co-occurrence graph with term co-occurrence frequency being set at 3. The α and β parameters for $cost(s_i)$ are set to 0.5.

Demonstration Scenarios. Consider a Java API java.util.HashMap as shown in Figure 2. To explore the caveats of this API, a programmer Tom starts by typing "java.util.HashMap" into the query box. API Caveat Explorer retrieves the API caveats and presents them on the user interface. Tom chooses to dig into the "*threading*" cluster by double clicking the corresponding block on *FoamTree* panel. Accordingly, the panels ③, ④, ⑤ synchronously display the five API caveats in this term community. He then chooses to dig deeper into the sub-cluster labeled "*concurrent*" (see Figure 3). Afterwards, he reads the API caveat "*Yes, HashMap is not synchronized so inserting elements from concurrent threads yields undefined results.*" and its original post on Stack Overflow. Finally, he discovers that there is a risk of sharing java.util.HashMap without proper synchronization.

Besides the *FoamTree* mode, API Caveat Explorer system also interactively presents the discovered API caveats using *Circles* mode and *Fisheye* mode. After exploring the non-synchronized feature of java.util.HashMap, Tom has a glance at the whole picture of API caveats in *Circles* mode. Furthermore, he finds that the "*threading*" community is associated with the "*keys*" community in *Circles* mode as shown in Figure 3. Subsequently, he clicks the node "*keys*" and reads the API caveat "*HashMap cannot store multiple values for the same key*" and its original post on Stack Overflow.

For java.util.HashMap, its long official documentation[6] mentions three API caveats related to *element order*, *multiple threads synchronization* and *comparable element*. Only the sentence for *multiple threads* is in bold text. Compared with the lengthy official documentation, API Caveat Explorer uses an interactive manner to surface much more hard-to-notice API caveats. By making these API caveats explicit, API Caveat Explorer provides an important reminder for novice programmers to avoid such mistakes. It is infeasible for API designers to take all these aspects into account when documenting API cookbooks, because these API caveats can only be accumulated in practice. On the contrary, API Caveat Explorer serves as a handbook of API caveats from practice.

REFERENCES

[1] D. M Blei, A. Y Ng, and M. I Jordan. 2003. Latent dirichlet allocation. *JMLR* 3 (2003), 993–1022.
[2] V. D Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.
[3] B. Dagenais and M. P Robillard. 2012. Recovering traceability links between an API and its learning resources. In *ICSE*. 47–57.
[4] G. Erkan and D. R Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *JAIR* 22 (2004), 457–479.
[5] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell. 1999. Summarizing text documents: sentence selection and evaluation metrics. In *SIGIR*. 121–128.
[6] J. Li, A. Sun, and Z. Xing. 2018. Learning to answer programming questions with software documentation through social context embedding. *Information Sciences* 448 (2018), 36–52.
[7] C.-Y. Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Proc. ACL-04 workshop*, Vol. 8.
[8] J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proc. Berkeley symposium on mathematical statistics and probability*, Vol. 1. 281–297.
[9] S. Subramanian, L. Inozemtseva, and R. Holmes. 2014. Live API documentation. In *ICSE*. 643–652.

[5] https://docs.oracle.com/javase/8/docs/api/overview-summary.html

[6] https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html