Searching API Usage Examples in Code Repositories with Sourcerer API Search

Sushil Bajracharya Joel Ossher Cristina Lopes
Donald Bren School of Information and Computer Sciences
University of California, Irvine
{sbajrach, jossher, lopes}@ics.uci.edu

Categories and Subject Descriptors

D.2 [**Software Engineering**]; H.3, H.5 [**Information Systems**]: Information Storage and Retrieval, Information Interfaces and Presentation

Keywords

Search Driven Development, Software Information Retrieval, API search, Exploratory Code Search, Search User Interface

ABSTRACT

We present Sourcerer API Search (SAS), a search interface to find API usage examples in large code repositories. SAS facilitates finding API usage examples by providing three unique features: (i) code snippets view for each result that shows the portions of code where APIs are used; (ii) Tag-cloud view of popular words to facilitate query reformulation, and (iii) filtering results using APIs to narrow search results. Furthermore, SAS uses a code index where each code entity is indexed with terms not only found in the entity but also in other entities having similar API usage. These features make SAS a novel search interface to find API usage examples in code repositories.

1. INTRODUCTION

Developers often learn to work with APIs by looking at existing examples [14]. Large code repositories host thousands of projects where many instances of API usage can be found. These instances of API usage can serve as examples from which developers can learn about using APIs to solve particular programming tasks. Existing Code search engines allow developers to search in large amount of underlying source code (for summary of code search engines, see [11]), but it is difficult to use these code search engines to find examples of API usage. Two factors contribute to this problem. First, source code compared to natural language text, has a scarcity of words that describe the entities defined and used in the code. This makes it more prone to the vocabulary mismatch problem [6]. Second, a code entity (such as a file, class or a method) returned from a code search engine often is too coarse-grained and large in size. When such entities are returned in search results, it if often difficult to locate the right position where one can see the relevant APIs being used. Code search engines often include snippets of code in the results page that allow users to see if the returned result is relevant, but such snippets often do not include code parts that use APIs of interest.

This paper presents Sourcerer API Search (SAS), a novel code search interface built on top of Sourcerer infrastructure, that makes searching for API usage examples in code repository easier for developers. In particular, SAS incorporates the following features: (i) Uses a code index that harvests more words for a code entity by seeding its index field with terms from similar entities, (ii) Generating and showing code snippets showing API usage, (iii) Aiding query reformulation by showing Tag-cloud of popular words for the current query, and (iv) Using popular APIs as filters to narrow down results.

In the rest of the paper we first illustrate how the features of SAS facilitate locating API usage examples by walking through a scenario where a developer is interested in finding examples for copying clipboard data in Eclipse framework. Second, we discuss how usage information available in SourcererDB [13], the program fact database in Sourcerer, is leveraged to implement the features used in SAS. Third, we discuss related work. Finally, we conclude by discussing future directions for SAS.

2. SOURCERER API SEARCH - SAS

To demonstrate the features of SAS, we will use a scenario where a developer wants to learn about the APIs to copy data from the clipboard in the eclipse framework [1]. Figure 1 shows the search interface of SAS. When a user enters a query such as "copy data from clipboard", SAS presents three major information as results:

- *Tag-cloud:* On top is a list of popular words found in the results in a Tag-cloud format. This Tag-cloud is generated by analyzing the names of the top code entities in the results (hits), the names of top APIs used by the hits, and the names of entities that have similar API usage as the hits.
- *Hits with Snippets:* Below Tag-clouds, on the right hand side is a list of code entities that match the query. The fully qualified name and a code snippet extracted from the source file of the code entity is shown as a hit. The code snippet is a collection of code lines preceded with a comment that indicates what API is used in the following line.
- *Top APIs:* Below the Tag-cloud and to the left is the list of popular (top) APIs that are used in the hits. This list serves

[1]Current version of SAS is tied to an underlying repository that contains source code of the Eclipse framework

SUITE '10, May 1 2010, Cape Town, South Africa

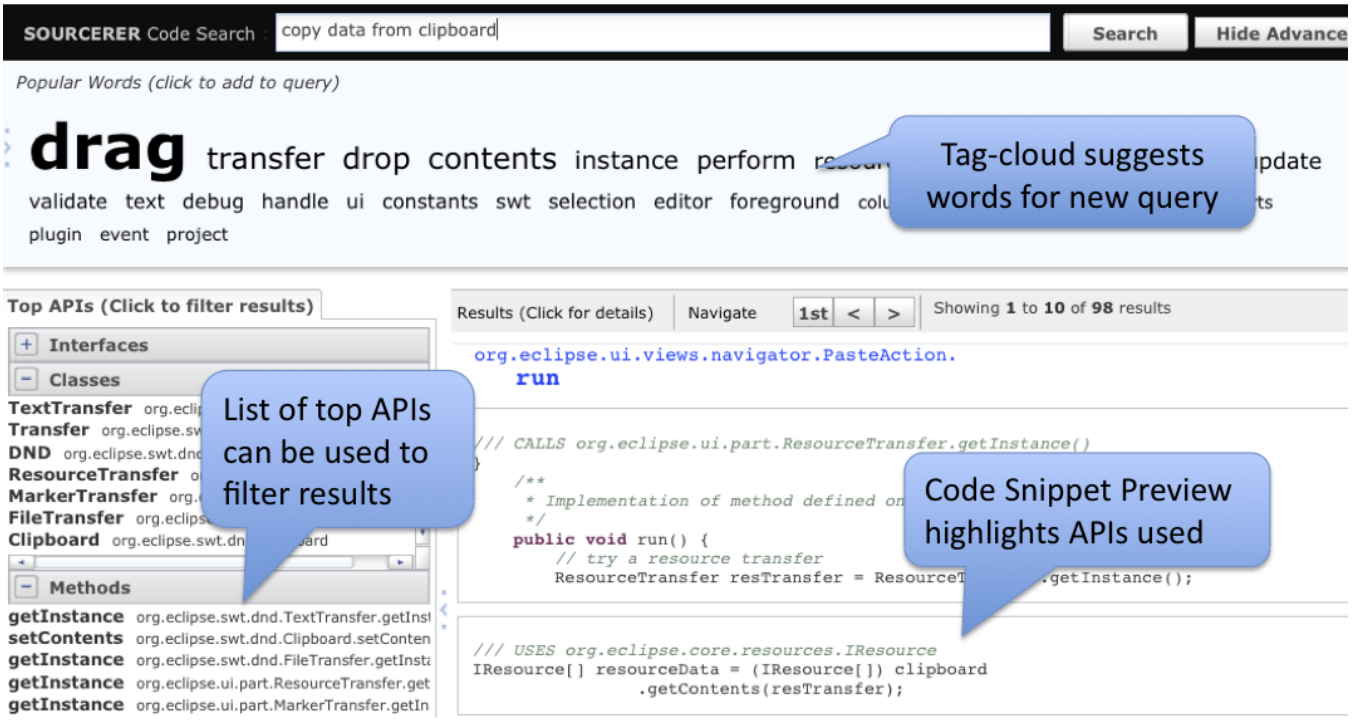

Figure 1: Sourcerer API Search - Features

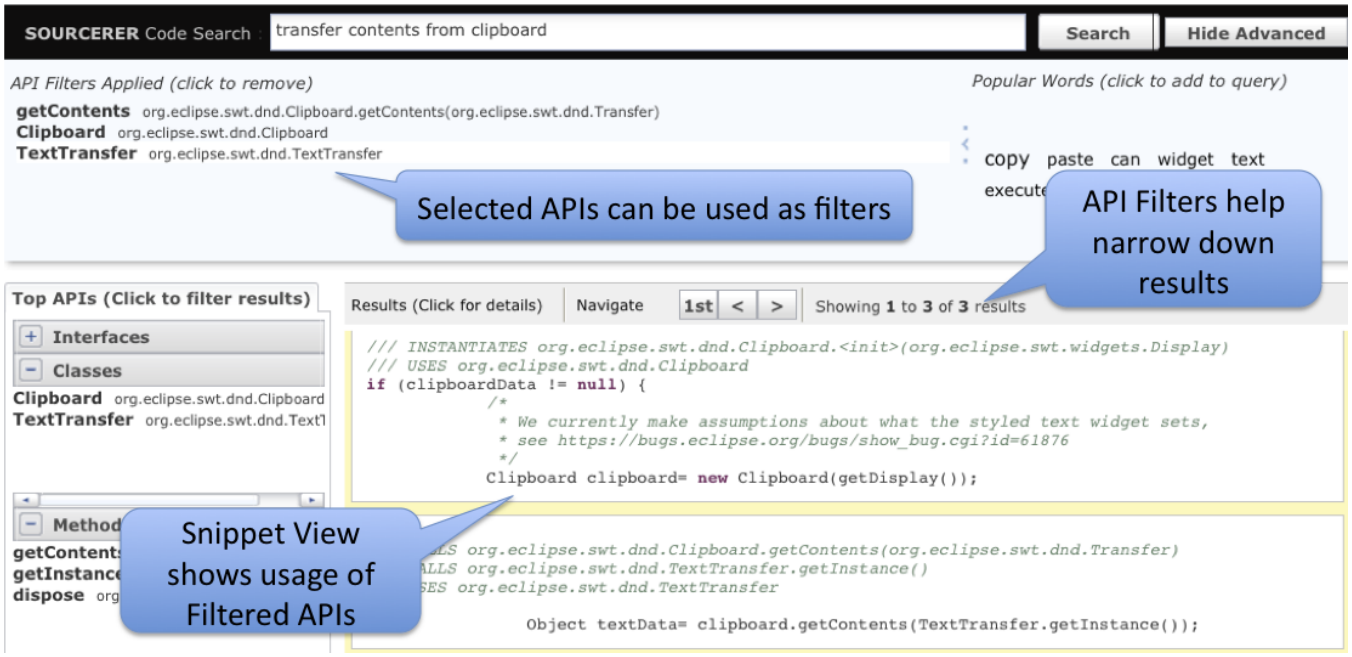

Figure 2: Using top APIs to filter results

two purpose: (i) it immediately shows the important APIs that are required to solve the programming task addressed by the query, and (ii) By clicking on a particular API element in the list, the search result can be filtered to all those entities that use that API element.

Upon getting a search result as shown in Figure 1, a user can modify the query to "transfer contents from clipboard", based on the popular words shown in the Tag-cloud. A user can further filter the results to those that use three top API elements `getContents(..)`, `Clipboard`, and `TextTransfer`. The result obtained after this in-

teraction is shown in Figure 2. It can be seen that the results have been narrowed down to 2 from 98 (as in Figure 1), and the code snippet shows the usage of all of the three selected APIs.

This scenario shows how SAS is different from existing code search engines. SAS focuses on finding API usage examples, and facilitates interaction that allows exploratory search for finding API usage examples in the underlying code repository. Such interaction has been shown to be effective in overcoming the vocabulary problem in search [9]. The prototype shown in the above figures is available for use, along with the source code. Please refer to [1] for further details.

3. IMPLEMENTATION

SAS has been implemented on top of the Sourcerer infrastructure. Sourcerer consists of set of tools and accompanying data to analyse and search large amount of source code. Details on the infrastructure is available in existing references [2, 13]. SAS sends the query given by a user to Sourcerer's search service. The search service is provided by Solr [2], a front end to the Lucene [3] index that is created from information stored in Sourcerer's file repository and SourcererDB [13].

3.1 Seeding Code Entity with Terms from Similar Entity

SAS uses a code index that includes a field for each code entity that contains terms extracted form other entities that are similar to it. Using such an index has shown to be effective in increasing retrieval performance in searching API usage examples [3]. Full details of this indexing technique is out of scope for this paper. We briefly present the idea here.

Computing Similar Entities based on Usage

A notion of similarity can be defined for code entities based on their common usage profile. For example, the Eclipse framework provides the method `getDebugOption` inside the class `Platform` located in the package `org.eclipse.core.runtime`. This method can be used to retrieve a value from a file storing a list of key/value pairs related to enabling options while running a Plugin in the debug mode. Some methods in the Eclipse framework that call this API have names such as: `configurePluginDebugOption`, `start`, `startupComplete`, `initDebugTracing`, `initializeJFace`, etc. Each of these methods is similar to the rest based on their functionality of retrieving a value for a debug option. Inspecting the names of these methods show that the names of the methods that use a common API method are made up of words having similar meaning; for example, 'start', 'init', 'startup', and 'initialize'. These examples suggest that using API usage similarity, we can harvest similar words to describe code entities.

Following the same approach as used by Bruch et al. in [4], usage information for each code entity is encoded as a binary vector. After obtaining this binary feature vector for each code entity, a neighborhood of similar users is obtained using various similarity measures for binary vectors. The FQNs of top 45 entities from such a neighborhood are extracted and stored back in SourcererDB for each code entity. This information is later retrieved during indexing a code entity to fill the respective usage similarity fields with the short names extracted from the FQNs of similar entities. For the purpose of SAS, we used the Tanimoto Coefficient [18] similarity as it turned out to be the best in terms of increasing retrieval performance in our evaluation [3].

[2]http://lucene.apache.org/solr/

[3]http://lucene.apache.org

3.2 Listing Top APIs

To list the top APIs, SAS fetches top-k hits from the results returned by the Sourcerer's search service (served via Solr server). And, for each hit in the top-k list, it queries SourcererDB for the used entities. Among these used entities, all entities that are used by more then 3 entities in the hits are included in the top APIs list [4]. This process is shown in Algorithm 1.

input : hits = top 'k' hits returned as search results; where, k = max_of(10, 10% of total hits)
output: top_used = list of top used entities

begin
  list_eid = all entity ids from hits;
  / getTopApis(..) selects top 5 non-JSL (Java Standard Library) entities of each type (Interface, Method, Constructor, Classes) from SourcererDB such that they are used by at least 3 entities in the hits */*
  top_used = getTopApis(hits);
end

Algorithm 1: Getting the list of top used entities

3.3 Snippet Extraction Details

The search results from Sourcerer's search service is a ranked result of code entities (hits) found in the index. The search tool also returns the total number of entities in the index that match the query. For each hit the corresponding 'entity_id' is available. Further details about the code entity can be queried from SourcererDB using the 'entity_id'. Snippet extraction proceeds in two steps. First, given a set of hits, a list of top used entities is generated as discussed earlier and shown in Figure 1. Second, a code snippet is extracted from a code entity, given the list of top used entities and the entity id. The algorithm for this process is shown in Algorithm 2.

3.4 Tag-cloud generation

To generate the Tag-cloud SAS uses a list of words extracted from the short names of the fully qualified names of code entities extracted from: (i) top-k entities in the result, (ii) top APIs used by top-k entities, and (iii) $30 * k$ similar entities, 30 for each entity in top-k hits. The weight of each word is proportional to the frequency of the word in the list, thus popular words appear bigger and earlier in the Tag-cloud.

4. RELATED WORK

The search interface of SAS is largely motivated by MICA [15], a search interface that allows developers to find APIs using Google. One contribution of SAS is that it demonstrates how an exploratory code search interface can be built solely based on information extracted in source code (unlike MICA). The technique to extract code snippets and top APIs is based on finding popular APIs among their users. This is similar to the work done in finding API hotspots [17]. Commenting code snippets with used APIs is motivated by the "Rationale" view in Strathcona that shows similar information about the code example it returns [10]. The idea of using API calls to improve retrieval has been explored before. SNIFF is a tool that improves retrieval by indexing client code with API documentation [5]; Exemplar is an application finding tool that uses API usage information to find relevant applications in a code repository [7]. Finally, the idea of providing interactive feedback using Tag-clouds

[4]We set 'k' to maximum of (10, 10% of total hits) after some experimentation

```
input  : eid = entity id, top_used = top used entities
output: snip = an annotated code snippet
begin
    snip = empty string;
    forall used_entity IN top_used do
        /* getPosition(..) returns the position in the file for
           entity_id using SourcererDB                          */
        position = getPosition(entity_id);
        /* createRationale(..) selects relation type and FQN of
           used entity and creates a rationale as a comment     */
        rationale = createRationale(used_entity,entity_id);
        /* extractFragment(..) extracts the surrounding
           expression in a code entity from position            */
        snip_fragment = extractFragment(entity_id,position);
        /* Below, append(a,b) returns a new string by
           appending string 'b' to 'a'. Condition is true if
           rationale and snip_fragment do not already exist in
           snip.                                                */
        if append(rationale,snip_fragment) ∉ snip then
            snip = append(snip,rationale);
            snip = append(snip,snip_fragment);
        end
    end
end
```

Algorithm 2: Snippet Extraction

and filtering based on APIs is motivated by the work in exploratory code search [12], in particular WordBars [9], and faceted-search [8].

5. FUTURE DIRECTION AND CONCLUSION

In this position paper we have demonstrated how usage information in code repositories can be leveraged to build an exploratory interface to find API usage examples. We implemented and demonstrated SAS as a prototype. Future directions for SAS includes a better usability study to assess its feasibility. We are interested in evaluating our hypothesis whether the exploratory interface SAS offers to find API usage examples can make developers more effective and efficient while they are searching for information on working with APIs. We believe features of SAS are complementary to existing alternatives such as Stratchona [10] and ParseWeb [16], especially because these tools are not designed to work with natural language queries. However, these tools provide better facilities for comprehending APIs, we look forward to investigate how features of SAS can complement such existing tools.

6. REFERENCES

[1] Sourcerer wiki page on sourcerer api search tool *http://wiki.github.com/sourcerer/Sourcerer/sas*.

[2] S. Bajracharya, J. Ossher, and C. Lopes. Sourcerer: An internet-scale software repository. In *First Intl. Workshop on Search Driven Development - Users, Infrastructure, Tools and Evaluation*. ICSE 2009, 2009.

[3] S. Bajracharya, J. Ossher, and C. Lopes. Leveraging usage similarity for effecive retrieval of examples in code repositories. *Manuscript Under Preparation*, 2010.

[4] M. Bruch, M. Monperrus, and M. Mezini. Learning from examples to improve code completion systems. In *Proceegings of FSE*, pages 213–222, Amsterdam, The Netherlands, 2009. ACM.

[5] S. Chatterjee, S. Juvekar, and K. Sen. SNIFF: A Search Engine for Java Using Free-Form Queries. In *Fundamental Approaches to Software Engineering*, pages 385–400. 2009.

[6] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30:964–971, 1987.

[7] M. Grechanik, K. M. Conroy, and K. A. Probst. Finding Relevant Applications for Prototyping. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 12. IEEE Computer Society, 2007.

[8] M. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, and K. Yee. Finding the flow in web site search. *Commun. ACM*, 45(9):42–49, 2002.

[9] O. Hoeber and X. D. Yang. Evaluating WordBars in exploratory web search scenarios. *Inf. Process. Manage.*, 44(2):485–510, 2008.

[10] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 117–125, New York, NY, USA, 2005. ACM Press.

[11] O. Hummel, W. Janjic, and C. Atkinson. Code conjurer: Pulling reusable software out of thin air. *IEEE Softw.*, 25(5):45–52, 2008.

[12] G. Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, 2006.

[13] J. Ossher, S. Bajracharya, and C. Lopes. SourcererDB: An aggregated repository of statically analyzed and cross-linked open source java projects. In *MSR 2009: 6th IEEE Working Conference on Mining Software Repositories*, 2009.

[14] M. B. Rosson and J. M. Carroll. The reuse of uses in smalltalk programming. *ACM Trans. Comput.-Hum. Interact.*, 3(3):219–253, 1996.

[15] J. Stylos and B. A. Myers. Mica: A Web-Search tool for finding API components and examples. In *Proceedings of the Visual Languages and Human-Centric Computing*, pages 195–202. IEEE Computer Society, 2006.

[16] S. Thummalapenta and T. Xie. Parseweb: a programmer assistant for reusing open source code on the web. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 204–213, Atlanta, Georgia, USA, 2007. ACM.

[17] S. Thummalapenta and T. Xie. SpotWeb: detecting framework hotspots via mining open source repositories on the web. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 109–112, Leipzig, Germany, 2008. ACM.

[18] P. Willett, J. M. Barnard, and G. M. Downs. Chemical Similarity Searching. *Journal of Chemical Information and Computer Sciences*, 38(6):983–996, Nov. 1998.