2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion

RACK: Code Search in the IDE using Crowdsourced Knowledge

Mohammad Masudur Rahman Chanchal K. Roy David Lo†
University of Saskatchewan, Canada, †Singapore Management University, Singapore
{masud.rahman, chanchal.roy}@usask.ca, davidlo@smu.edu.sg

***Abstract*—Traditional code search engines often do not perform well with natural language queries since they mostly apply keyword matching. These engines thus require carefully designed queries containing information about programming APIs for code search. Unfortunately, existing studies suggest that preparing an effective query for code search is both challenging and time consuming for the developers. In this paper, we propose a novel code search tool–RACK–that returns relevant source code for a given code search query written in natural language text. The tool first translates the query into a list of relevant API classes by mining keyword-API associations from the crowdsourced knowledge of Stack Overflow, and then applies the reformulated query to GitHub code search API for collecting relevant results. Once a query related to a programming task is submitted, the tool automatically mines relevant code snippets from thousands of open-source projects, and displays them as a ranked list within the context of the developer's programming environment–the IDE. Tool page: http://www.usask.ca/∼masud.rahman/rack**

***Index Terms*—Code search, query reformulation, keyword-API association, crowdsourced knowledge, Stack Overflow**

I. Introduction

Studies show that software developers on average spend about 19% of their development time in web search where they mostly look for relevant code snippets for their programming tasks [1]. Code search engines–*Open Hub, Koders, GitHub search* and *Krugle*–index thousands of open source projects which are a potential source for such snippets [7]. Unfortunately, preparing an effective query for code search containing information about relevant APIs is not only a challenging task but also is time-consuming for the developers [1, 5]. Previous study also reported that on average, developers performed poorly in coming up with good search terms regardless of their experience levels [5]. Thus, a tool that automatically translates a natural language query from the developer into a set of relevant API classes or methods (i.e., search-engine friendly query) and then returns relevant source code snippets, can greatly assist the developers in their tasks. Our paper addresses this research problem, and provides automatic tool support both in preparing search queries and in performing code search conveniently.

Existing studies accept one or more natural language queries, and return relevant API classes and methods by analyzing feature request history and API documentations [10], API invocation graphs [2], library usage patterns, code surfing behaviour of the developers and API invocation chains [7]. Although these techniques perform well in different working contexts, they share a set of limitations and fall short to address our research problem. First, each of these techniques [2, 7, 10] exploits textual similarity measure (e.g., Dice's coefficients [2]) for candidate API selection. This warrants that the search query should be carefully prepared, and it should contain keywords similar to the API names. In other words, the developer should possess a certain level of experience on the target APIs to actually use those techniques. Second, API names and search queries are generally provided by different developers who may use different vocabularies to convey the same concept. Concept/feature/concern location community have termed it as *vocabulary mismatch problem* [4]. Textual similarity based techniques often suffer from this problem. Hence, the performance of these techniques is not only limited but also subject to the identifier naming practices adopted in the codebase under study.

In this paper, we propose a novel code search tool–*RACK*–that accepts an unstructured natural language query (i.e., does not require API information) from a developer as input and returns relevant code snippets as output from thousands of open source projects. The tool first captures the developer's intent for code search from two working contexts (e.g., code comments) within the IDE as a query, translates the query into relevant API classes automatically, and then collects the relevant code examples from GitHub search API by applying them. While each question in Stack Overflow Q & A site summarizes a programming problem/task, the corresponding answers often suggest appropriate APIs that solve the problem. We thus mine thousands of questions and corresponding accepted answers from Stack Overflow, and translate the natural language query (i.e., programming task) into relevant API classes by exploiting the keyword-API associations from Stack Overflow. Thus, the tool works both as a query recommender and as a code search engine. We package our solution as an Eclipse IDE plug-in that allows the developers to perform code search within the IDE, and thus, they can avoid the annoying context-switching issue. To summarize, our tool provides the following features to support the developers in code search:

(a) the proposed tool automatically translates an unstructured natural language query referring to a programming task into relevant API classes for the task.
(b) determines relevance of the returned API classes based on keyword-API associations mined from thousands of programming questions and solutions of Stack Overflow.
(c) mitigates the vocabulary mismatch problem faced by existing techniques and traditional code search engines.
(d) integrates GitHub search API into the IDE for IDE-based code search and convenient result display.

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.11

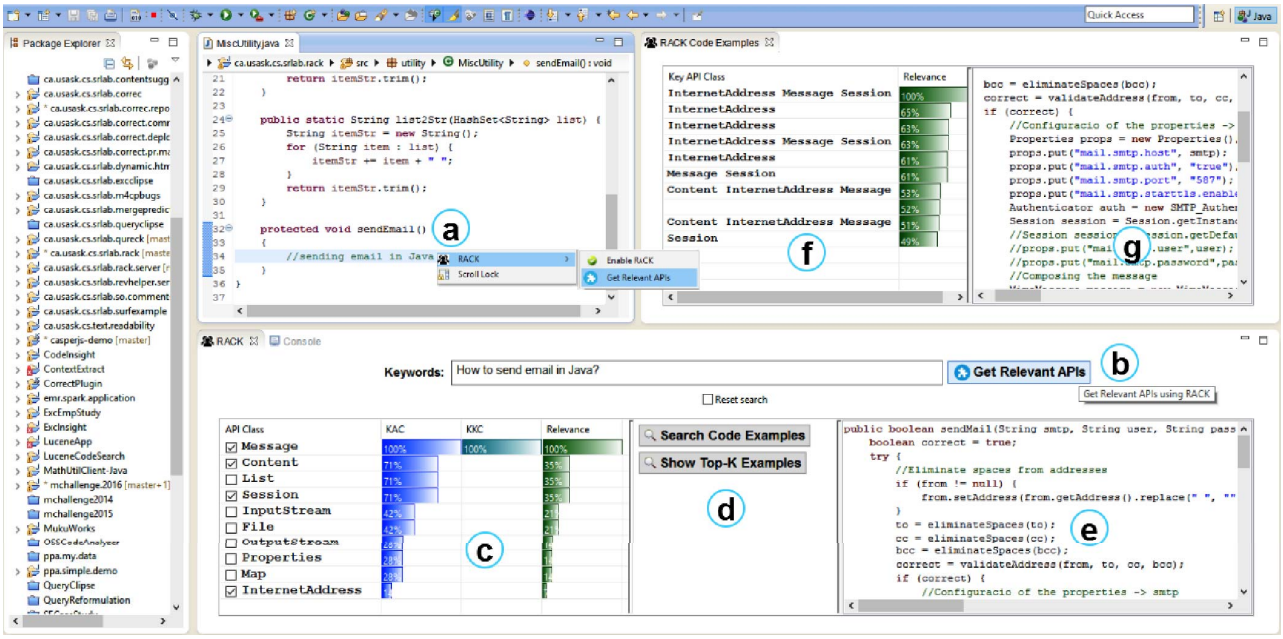

Fig. 1. User Interface of RACK

(e) offers meaningful relevance insights for both search queries and search results unlike any traditional search.

While this paper focuses on tool aspect of our approach, we refer the readers to the original paper [8] for further details.

II. RACK

Fig. 1 shows the user interface of *RACK*, where we contribute in (b)–(c) query suggestion panel, (d)–(e) code search panel, and (f)–(g) result panel of the interface. This section discusses different technical features provided by our tool.

(1) Automatic Suggestion of Code Search Queries: RACK automatically suggests relevant keywords (i.e., API classes) for code search given that preparing an effective query for a programming task is a significant challenge [5, 6]. Our tool overcomes this challenge by mining thousands of programming problems and their corresponding solutions from Stack Overflow Q & A site. It captures a developer's intent for code search from various working contexts within the IDE, and suggests a list of appropriate keywords for code search with meaningful insights (i.e., relevance scores).

(i) Working Contexts: RACK captures natural language queries (i.e., developer's intents) for code search from two working contexts of a developer– *source code comment* and *traditional search box*. Once the developer intends to accomplish a programming task by stating in the header comment of a method, our tool captures the comment as an initial query for reformulation (e.g., Fig. 1-(a)). In the second case, the developer provides an initial query written using unstructured natural language texts, and RACK captures the query from the traditional search box (Fig. 1-(b)) for relevant API suggestion.

(ii) Mining of Relevant API Classes: In Stack Overflow, users often submit questions focusing programming tasks (e.g., "How can I generate MD5 hash?"), and the corresponding answers suggest relevant APIs (e.g., `MessageDigest`) for accomplishing those tasks. RACK accesses a database of 344K such questions and answers, learns keyword–API associations, and then suggests relevant API classes for a given task.

(iii) API Suggestion and Query Reformulation: Once the natural language query (i.e., initial query) is submitted, RACK suggests the Top-10 relevant API classes for the task in the query (Fig. 1-(c)). Not only the suggestions are provided as a ranked list but also our tool explains why a particular API is relevant by visualizing three meaningful scores– Keyword–API Co-occurrence (KAC), Keyword–Keyword Coherence (KKC), and Overall Relevance [8]. Being equipped with such ranking and insights, a developer can easily choose appropriate APIs by marking them checked and initiate the code search.

(2) IDE-Based Code Search: RACK not only provides an IDE-based code search feature but also assists the developer in result analysis with a customized view. It provides two flexible code search options and displays the results with meaningful insights (i.e., relevance scores) within the IDE.

(i) Code Search Options and Backend: RACK provides two options–Top-1 search and Top-K search–for performing code search in the IDE (Fig. 1-(d)). Once the relevant API classes are suggested (by the tool) and appropriate classes (i.e., search keywords) are chosen by the developer, the tool returns the topmost relevant code snippet from thousands of open source projects of four large organizations–*Apache, Eclipse, Google* and *Facebook*. We integrate GitHub code search API in the backend for collecting the relevant source code files from which the most relevant method body is extracted using Abstract Syntax Tree (AST) parsing and textual similarity analysis with the query. In the second case, RACK returns the Top-K (e.g., $K = 10$) code snippets based on their relevance for further analysis by the developer. One can also reset the whole process by checking the check box provided by the tool.

TABLE I
SUGGESTED API CLASSES FOR THE USE CASE

API	KAC	KKC	Relevance	API	KAC	KKC	Relevance
File	0.60	1.00	1.00	Element	0.60	0.46	0.66
Document	1.00	0.46	0.91	Jsoup	0.40	0.00	0.25
List	0.90	0.22	0.70	Elements	0.20	0.00	0.19

(ii) Mitigation of Vocabulary Mismatch Issue: Textual similarity based search techniques (e.g., Vector Space Model) generally suffer from this issue when unstructured natural language queries are used for code search [3, 4]. Since RACK translates the initial search query into relevant API classes that come from standard libraries or development toolkits (i.e., from a single vocabulary), such issue is mitigated.

(iii) Result Display and Insights: RACK not only shows the code search results as a meaningful ranked list (i.e., with relevance insights) but also adds an in-line source code viewer for detailed analysis of the results (Fig. 1-(e)–(g)). Each result from the list is annotated using the matched keywords from the query which provides additional intuition about its relevance. The code viewer is enabled with *syntax highlighting* which ensures a convenient analysis of the code by the developer.

(3) Performance Optimization: While the query reformulation step requires relational database access, the code search step involves GitHub API access and significant static analysis of the source code. In both steps, RACK applies Java multi-threading for optimized computation and response time. To date, our reformulation takes $\leq$10 seconds and the search takes $\leq$2 seconds on average which are close to real time.

(4) Seamless Integration and Dynamic Corpus: RACK adopts a client-server architecture where the Eclipse IDE plug-in is the client module, and the server module (i.e., query reformulation engine) is hosted as a web service. That is, any tool capable of making HTTP calls can consume our query reformulation service, which demonstrates RACK's modularity. On the other hand, the use of GitHub API ensures that RACK always returns relevant code from an automatically evolving and carefully indexed large source code corpus.

III. A USE CASE SCENARIO

By means of a use case scenario, we attempt to explain how *RACK* can help a software developer in accomplishing a programming task within the IDE.

Suppose a developer, Alice, is attempting to develop a Java application that parses an HTML page (e.g., Yahoo! finance page), and extracts certain items of her interest (e.g., stock price). However, she lacks necessary experience and thus is looking for a working code example that performs the same or similar task. She formulates a query–*"parsing html in Java"*, and submits to a web search engine (e.g., Google). The search engine leads her to a list of programming Q & A pages and API documentations. Now, she needs to go through the pages carefully containing a large body of texts. While these pages might be useful for improving her knowledge on parsing, choosing relevant code examples from them is not only a time consuming but also a non-trivial job. She also submits the same natural language query to a code search engine (e.g., GitHub), but the returned results were not promising. In short,

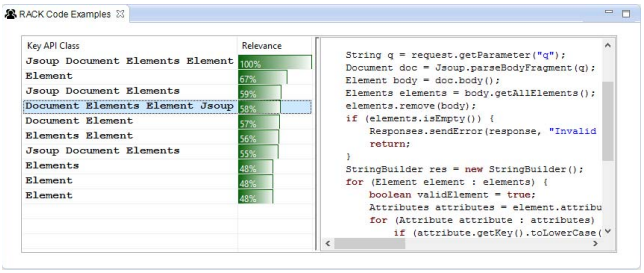

Fig. 2. Top-10 code search results for the use case

she (1) fails to collect a succinct and working code example comfortably from the web search results due to the noise in the content, (2) does not get a relevant result from the code search engine due to its inherent limitation–vocabulary mismatch issue between the query and source code, (3) finds the display of neither web search results nor code search results helpful for post-search analysis (i.e., trying out examples).

Now, let us assume that Alice has installed *RACK* in her IDE, and she encounters the same programming challenge. Our tool captures her natural language query from the code comment (e.g., Fig. 1-(a)), and automatically suggests a ranked list of relevant API classes along with three relevance insights (i.e., KAC, KKC and Relevance) for the task. Table I shows the Top-6 APIs suggested by RACK. Among them four (i.e., 67%) classes–`Document`, `Element`, `Jsoup` and `Elements`–are related to HTML parsing. She can play along with the top API classes, reformulate the initial query, and instantly try out the working code examples returned by the reformulated query. Existing study reported that developers frequently experiment with and learn from working code examples [1]. Fig. 2 shows the Top-10 relevant code snippets returned by RACK for this use case which are mined from thousands of open source projects using GitHub code search API. Not only our tool provides the relevance estimate for each result but also it annotates them with matched keywords and adds an in-line source code viewer. Such information and feature are likely to assist one in analyzing the code results more conveniently.

Thus, RACK (1) provides Alice one or more succinct and relevant code example(s) without much effort or time spent (i.e., 10-15 seconds), (2) overcomes the vocabulary mismatch issue of a traditional code search engine through effective query reformulation (i.e., relevant API classes), and (3) displays the results with meaningful insights and convenient viewing panel. In short, our tool does all the heavy lifting on behalf of Alice and provides a better alternative than the traditional means for code search and her problem solving.

IV. WORKING METHODOLOGY

Fig. 3 shows the schematic diagram of our proposed tool. This section discusses the internal structures and working methodologies of the tool in brief, while we refer the readers to the original paper [8] for details.

Construction of Keyword-API Mapping Database: We first construct our keyword-API mapping database by carefully analyzing 344K programming questions and corresponding accepted answers (i.e., solutions) from Stack Overflow Q & A

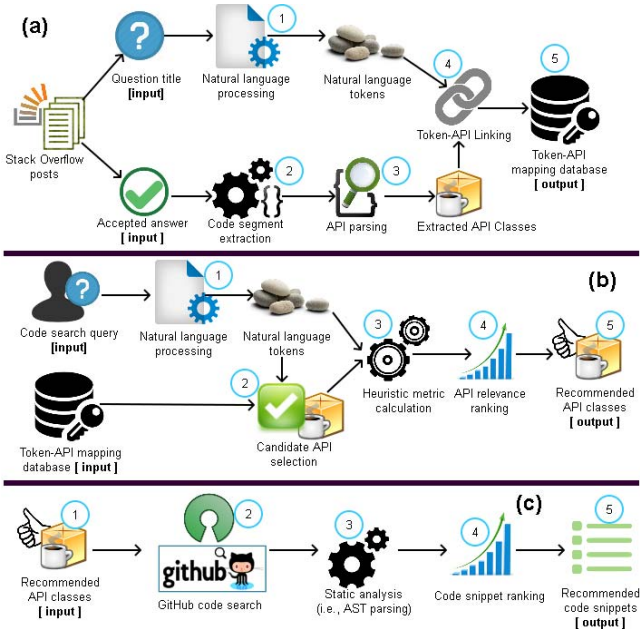

Fig. 3. Schematic diagram of RACK – (a) Construction of keyword-API mapping database, (a) Reformulation of a natural language query, and (c) Code snippet search

site. The keywords are collected from the question titles using natural language preprocessing whereas the API classes are extracted from the answers through island parsing (i.e., Steps 1–3, Fig. 3-(a)). Then we capture the inherent associations between the keywords and the API classes from each question-answer pair, and construct the keyword-API mapping database (Steps 4-5, Fig. 3-(a)). RACK accesses this database for reformulating a natural language query.

Query Reformulation: Once an initial query written in unstructured natural language is submitted to RACK, the query is sanitized through natural language preprocessing (i.e., stop word removal, token splitting, stemming) and converted into a vector of keywords. Then those keywords are used to collect the candidate API classes from the mapping database using two heuristics–KAC and KKC (i.e., Steps 1–3, Fig. 3-(b)). Then the candidates are ranked based on their likelihood (i.e., derived from KAC) and coherence (i.e., derived from KKC) with the keywords. Finally, the tool returns a ranked list of relevant API classes (with relevance estimates) as a reformulation of the initial query (i.e., Steps 4, 5, Fig. 3-(b)).

Code Search in the IDE: Once a reformulated query containing appropriate/relevant API classes is submitted to RACK, it uses GitHub search API and collects relevant source code files from thousands of open source projects hosted by four large organizations–Apache, Eclipse, Google and Facebook. Given that developers are often interested in trying out the code snippets performing a particular task, we parse all the methods from each source file using AST-based parsing (e.g., *Javaparser* library) (i.e., Steps 1–3, Fig. 3-(c)). GitHub API returns a relevance score for each result file which we combine with the textual similarity scores (with the search query) of all the methods extracted from that file. This combination provides a combined relevance for each code snippet (i.e., method), and RACK finally returns a ranked list of relevant code snippets within the IDE (i.e., Steps 4, 5, Fig. 3-(c)).

V. Performance

Since our original paper [8] claims main contributions in the API recommendation for query reformulation, that part of RACK was rigorously evaluated and validated. To evaluate the API suggestion performance, we conduct experiments using 150 code search queries randomly chosen from three programming tutorial sites–*KodeJava, Java2s* and *JavaDB*. The evaluation shows that RACK was able to suggest at least one relevant API class for 79% of the queries within the Top-10 API suggestions, which is highly promising according to the literature. Comparison with the state-of-the-art–Thung et al. [10]–not only validated our performance but also confirmed the superiority of RACK in relevant API suggestion. Since our reformulated queries contain gold set API classes and we exploit GitHub API for code search, our queries are also likely to return relevant code snippets given that GitHub applies keyword matching in source code search.

VI. Conclusion & Future Work

To summarize, we propose a novel IDE-based code search tool–*RACK*–that returns relevant code snippets for natural language queries unlike the traditional code search engines. It exploits crowdsourced knowledge from Stack Overflow for query reformulation (details in the original paper [8]), and then applies the reformulated queries to collecting relevant code from GitHub search API. In future, we plan to conduct an exhausted user study with the tool involving prospective participants. We also plan to apply the inherent mapping between keywords and API classes mined from Stack Overflow posts to several other software maintenance activities such as concept location, bug localization and source code re-documentation.

Acknowledgement

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

References

[1] J. Brandt, P.J. Guo, J. Lewenstein, M. Dontcheva, and S.R. Klemmer. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proc. SIGCHI*, pages 1589–1598, 2009.
[2] W. Chan, H. Cheng, and D. Lo. Searching Connected API Subgraph via Text Phrases. In *Proc. FSE*, pages 10:1–10:11, 2012.
[3] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The Vocabulary Problem in Human-system Communication. *Commun. ACM*, 30(11):964–971, 1987.
[4] S. Haiduc and A. Marcus. On the Effect of the Query in IR-based Concept Location. In *Proc. ICPC*, pages 234–237, June 2011.
[5] K. Kevic and T. Fritz. Automatic Search Term Identification for Change Tasks. In *Proc. ICSE*, pages 468–471, 2014.
[6] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich. Feature Location via Information Retrieval Based Filtering of a Single Scenario Execution Trace. In *Proc. ASE*, pages 234–243, 2007.
[7] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu. Portfolio: Finding Relevant Functions and their Usage. In *Proc. ICSE*, pages 111–120, 2011.
[8] M. M. Rahman, C. K. Roy, and D. Lo. RACK: Automatic API Recommendation using Crowdsourced Knowledge. In *Proc. SANER*, pages 349–359, 2016.
[9] S. E. Sim, M. Umarji, S. Ratanotayanon, and C. V. Lopes. How Well Do Search Engines Support Code Retrieval on the Web? *TOSEM*, 21(1):4:1–4:25, 2011.
[10] F. Thung, S. Wang, D. Lo, and J. Lawall. Automatic Recommendation of API Methods from Feature Requests. In *Proc. ASE*, pages 290–300, 2013.