8-2016

API recommendation system for software development

FERDIAN THUNG
Singapore Management University, ferdiant.2013@phdis.smu.edu.sg

Citation

FERDIAN THUNG. API recommendation system for software development. (2016). *ASE 2016: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering: Singapore, September 3-7, 2016*. 896-899.
Available at: https://ink.library.smu.edu.sg/sis_research/3620

ASE 2016: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering:
Singapore, September 3-7, 2016. pp. 896-899.
http://doi.org/10.1145/2970276.2975940

API Recommendation System for Software Development

Ferdian Thung
School of Information Systems
Singapore Management University
ferdiant.2013@smu.edu.sg

ABSTRACT

Nowadays, software developers often utilize existing third party libraries and make use of Application Programming Interface (API) to develop a software. However, it is not always obvious which library to use or whether the chosen library will play well with other libraries in the system. Furthermore, developers need to spend some time to understand the API to the point that they can freely use the API methods and putting the right parameters inside them. In this work, I plan to automatically recommend relevant APIs to developers. This API recommendation can be divided into multiple stages. First, we can recommend relevant libraries provided a given task to complete. Second, we can recommend relevant API methods that developer can use to program the required task. Third, we can recommend correct parameters for a given method according to its context. Last but not least, we can recommend how different API methods can be combined to achieve a given task.

In effort to realize this API recommendation system, I have published two related papers. The first one deals with recommending additional relevant API libraries given known useful API libraries for the target program. This system can achieve recall rate@5 of 0.852 and recall rate@10 of 0.894 in recommending additional relevant libraries. The second one deals with recommending relevant API methods a given target API and a textual description of the task. This system can achieve recall-rate@5 of 0.690 and recall-rate@10 of 0.779. The results for both system indicate that the systems are useful and capable in recommending the right API/library reasonably well. Currently, I am working on another system which can recommend web APIs (i.e., libraries) given a description of the task. I am also working on a system that recommends correct parameters given an API method. In the future, I also plan to realize API composition recommendation for the given task.

CCS Concepts

•Software and its engineering → Software libraries and repositories; •Information systems → Recommender systems;

Keywords

API; Library; Recommendation System

1. INTRODUCTION

Gone are the days when developers have to build a software from scratch. Proliferation of third party libraries makes software development task faster and easier than ever before. In fact, it has been an integral part or software development [2, 9, 16]. In one of my paper [22], I have also investigated a number of GitHub projects with substantial size and found that 93.3% of them use third party libraries. Indeed, using third party libraries allow developers to save time since they do not need to reinvent the wheel. Instead, they can focus on the specific task at hand. Third party libraries are also tested; thus, they reduce the likelihood of creating bugs.

Although third party libraries are readily available, finding relevant libraries and using them are not always straightforward. Developers need to find relevant libraries among thousands and manually check whether the library functionality is a match with their requirement. This manual checking might involve a long process involving finding relevant methods in the API and understanding how to use them. Comprehending the structure of API and how to fill method parameters properly are among the things that developers need to learn. Clearly, this process can be time consuming and the existence of an API recommendation system would help developers a great deal in doing their job.

In order to help developers find and use API libraries, I plan to build an API recommendation system for software development. I divide API recommendation into multiple stages. At first, we can recommend which libraries to pick. Second, we recommend the method to use. Third, we recommend how to fill parameters. Lastly, we recommend how to compose the API methods. I have published two papers related to this line of work.

The first paper recommends API library given a known to be useful or existing libraries in the system [22]. This recommendation system can help developers to find additional relevant APIs. At this point, it was assumed that developers have some idea on APIs to use. The recommendation system is built by combining association rule mining

and collaborative filtering techniques. Experiments on the approach shows that it can achieve recall rate@5 of 0.852 and recall rate@10 of 0.894 for recommending additional libraries.

The second paper recommends API methods given a target library and a textual description of the task [21]. This recommendation system can help developers to find relevant API methods in the given library that can be used to realize the task. The recommendation system is built by combining information retrieval (IR) and collaborative filtering techniques. Experiment on the approach shows that it can achieve recall-rate@5 of 0.690 and recall-rate@10 of 0.779 for recommending API methods.

To continue the work on API recommendation system, I am currently in progress of building web API recommendation. This is essentially also a library recommendation system that takes as input a textual description of the given task. For this system, I have built a personalized ranking model to rank known libraries based on their expected relevancy given the description. Other than this system, I am also building an API method parameter recommendation system. For this system, I am planning to rank parameters based on the surrounding context of the code in which the method is to be inserted. Unlike the library recommendation system, this recommendation system is still in an early progress and thus still under a lot of exploration. In the future, I am also planning to create an API composition recommendation system. This system would be able to recommend whether two or more API methods can be composed and how to compose them.

The structure of this paper is as follows. In Section 2, I describe my published proposed works in API recommendation. In Section 3, I explain my ongoing works in the area. I then discuss related works in Section 4. I then conclude and talk about future work in Section 5.

2. PUBLISHED WORK

2.1 Library Recommendation

Figure 1 shows the proposed library recommendation framework. It was named as *LibRec* [22]. It consists of two main components: $LibRec^{RULE}$ and $LibRec^{COLLAB}$. $LibRec^{RULE}$ makes use library usage patterns to recommend libraries. The patterns are mined using association rule mining technique. $LibRec^{COLLAB}$ makes use a nearest-neighbor-based collaborative filtering approach to recommend libraries that are used by similar projects. In the training phase, our system extracts model from a training dataset (*TrainingProjects*). *TrainingProjects* is a set of projects that contains information of third-party libraries that they use. Model extraction

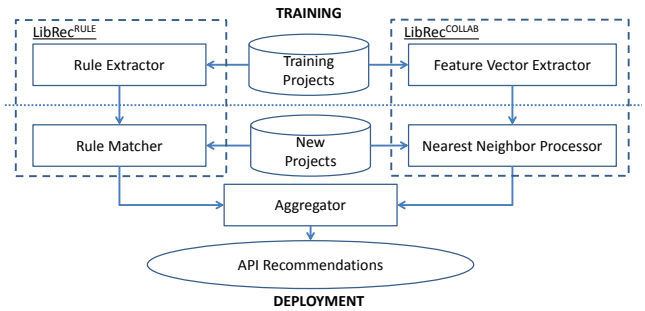

Figure 1: Architecture of *LibRec*

is specifically performed by the following sub-components:

1. *RuleExtractor*, capture library usage patterns from *Training Projects* by mining association rules.

2. *FeatureVectorExtractor*, extracts a feature vector from the set of libraries that the project uses. Each feature dimension represents a specific library. A value of 1 indicates that the library is used by the project and 0 otherwise.

Given the association rules and feature vectors model, in the deployment phase, our system can recommend libraries to new projects (*NewProjects*). *NewProjects* is a set of new projects that contains information of third-party libraries that they have used. Library recommendation is performed by the following sub-components:

1. *RuleMatcher*, takes a new project in $NewProjects$ and the association rule model as inputs. It then find the rules that match the libraries used in the project. The recommendation is made based on the post-conditions of the matching rules.

2. *NearestNeighborProcessor*, takes a new project in $NewProjects$ and the feature vectors model as inputs. It then calculates the distance between each of feature vectors in the model and the feature vector generated from the new project. Based on this distance, the nearest neighbors are identified. Most recommended libraries in this component will be the one that is used by the highest number of nearest neighbors. This whole process is based on collaborative filtering approach.

Both *RuleMatcher* and *NearestNeighborProcessor* would give a score for each library in the database. The *Aggregator* component computes final recommendation score by linearly combining the two scores. Libraries having the highest scores will be recommended.

2.2 API Method Recommendation

The proposed API method recommendation framework is shown in Figure 2 [21]. This framework contains three major components: *History Based Recommender*, *Description Based Recommender*, and *Integrator*.

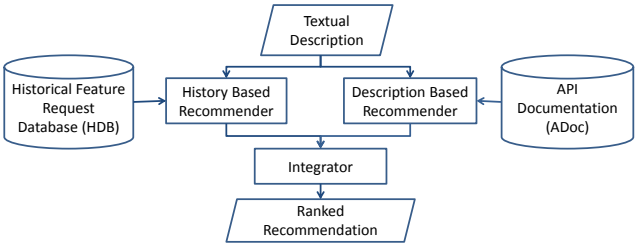

Figure 2: API Method Recommendation Framework

History Based Recommender takes as input the description of a new feature request, i.e. a task, (*Textual Description*) and a historical database containing past feature requests (*Historical Feature Request Database (HDB)*). The recommender finds feature requests in the historical database that are the closest to the new feature request. The methods that are used to implement the closest feature requests

are ranked higher. This recommendation is based on collaborative filtering approach. *Description Based Recommender* takes as input the description of the new feature request (*Textual Description*) and the API libraries documentation(*API Documentations (ADoc)*). The recommender computes the similarity of the feature request description with each method description in the API documentation. Methods whose textual descriptions are the most similar with the new feature request description are more recommended. This recommendation is based on information retrieval approach. *History Based Recommender* and *Description Based Recommender* output recommendation scores for each API method. *Integrator* combines the recommendation scores and outputs a list of API methods ranked by the final combination score. API method with the highest score is first recommended to the developer.

3. ONGOING WORK

3.1 Web API Recommendation

The architecture of currently developed web API recommendation system `WebAPIRec`, is shown in Figure 3. It takes as input a description of new project, a set of completed projects, and a set of API descriptions. The new project description contains a textual description and set of keywords for the project. For each completed projects, other than description of the project (i.e., text and keywords), actual APIs that are used in the project are also recorded. Similar like project description, API description also contain a textual description and a set of keywords for the API. `WebAPIRec` analyzes these inputs and finally produces a ranked list of APIs to be recommended to the target project.

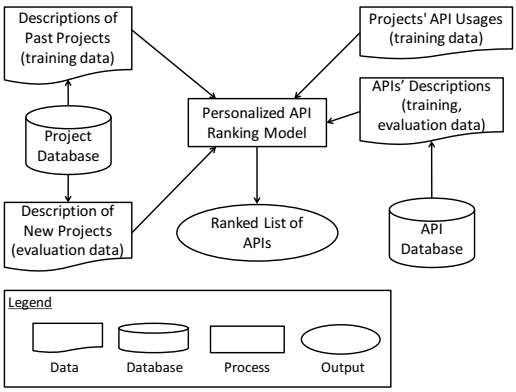

Figure 3: Architecture of `WebAPIRec`

`WebAPIRec` consists of two phases: *training phase* and *deployment phase*. In training phase, `WebAPIRec` converts the input project descriptions to feature vectors. `WebAPIRec` first identifies nouns from the textual descriptions using Stanford Parts-of-Speech (POS) tagger. These words carry more meaning than other kinds of words, c.f. [5, 18]. `WebAPIRec` combines the nouns with the keywords, remove stop words, stem each word, and finally construct a feature vector. The same process is executed to convert an API's textual description into a feature vector. These project and API feature vectors are put to a set of training triples (p, a, a') for the personalized ranking model. In a triple (p, a, a'), p is the feature vector of a project, a is the feature vector of an API in p, and a' is the feature vector of an API not in p. From this data, the personalized API ranking model is learned. The personalized API ranking model is built as a variant of *ranking support vector machine* (RankSVM) [8].

In deployment phase, similar to the *training phase*, `WebAPIRec` constructs feature vectors from a new project. Leveraging the personalized API ranking model, `WebAPIRec` computes the relevancy of the APIs in database and sort them in descending order based on the recommendation scores. To evaluate `WebAPIRec`, the recommendation is checked against the ground truth.

3.2 API Method Parameter Recommendation

This direction is still on the early stage. The closest works in this area is done by Zhang et al. [23] and Asaduzzaman et al. [1]. Asaduzzaman et al. is the state of the art and it has been shown that their approach beats Zhang et al.'s approach. Their approach leverages parameter usage context using a source code localness property and static type analysis to recommend method parameters. In this direction, I plan to develop a better approach that beats Asaduzzaman et al.'s approach. One possible improvement is to use a better ranking method. I am experimenting with different ranking methods to find an approach that suitable for ranking API parameters. One promising direction is to learn from historical API parameter usage. Another possible improvement is to develop better features for identifying relevant parameters. For example, two parameters might be more likely to appear together. Thus, knowing one correct parameter could make the recommendation for the other parameters better. For evaluation, I will check whether it matches the ground truth.

4. RELATED WORK

Mandelin *et al.* proposed a tool to recommend a code snippet that converts an object of one type to another [11]. Their work was extended by Thummalapenta and Xie [20] who used a code search engine to solve the same problem. This approach queries a code search engine (i.e., Google Code) to return a set of code examples. Bruch *et al.* recommended method calls for code auto-completion based on the context of the code that a developer is working on [4]. Robbes and Lanza proposed a technique to improve code auto-completion by using recorded program history [14]. Hindle *et al.* proposed a code auto-completion feature using a statistical language model [7]. Chan *et al.* proposed an approach to recommend API methods given textual phrases [6]. Moreno *et al.* proposed MUSE to recommend code examples by extracting concrete method usages [13].

Robillard *et al.* proposed Suade, which takes as input a set of program elements of interest and outputs another set of program elements that are likely to be interesting to developers [15]. Their approach was extended by Saul *et al.*, which employs an algorithm named FRAN to recommend a set of methods that are relevant to a target method by performing random walk on a call graph [17]. Long *et al.* proposed Altair which extends Suade and Fran by recommending methods to a target method based on variables that are shared among them [10]. Zhang *et al.* extended the previous approaches by making use of a call graph that is enhanced with control flow information [24]. Zimmermann *et al.* use association rule mining to infer program elements that also need to also be changed, when a set of program elements are changed [25].

Teyton *et al.* proposed an approach that creates a library

migration graph by analyzing library migrations performed by a large number of projects [19]. McMillan *et al.* proposed an approach to recommend software packages for rapid prototyping given product descriptions [12]. The approach first recommends feature to the user and use the user selected features to recommend the relevant software modules. Bianchini *et al.* presented a framework for different web API search patterns [3]. It supports recommending web API for a new mashup, adding a web API to a mashup, and replacing an existing web API in a mashup.

5. CONCLUSION AND FUTURE WORK

I have developed two systems for API recommendation. One recommends API libraries while the other recommends API methods. I currently have two ongoing works. One of the works is essentially recommending API libraries. However, instead of needing some known libraries, this system only requires a textual description of the task. The other ongoing work is dealing with a problem of recommending parameters for an API method. I am also planning to do API composition recommendation work in the future. The target of such system is the ability to automatically arrange the correct order of API and ideally synthesize the relevant *glue codes* to make the APIs work together. Eventually, I want to deploy the systems to get feedback from developers.

6. ACKNOWLEDGEMENT

My dissertation research would not have been possible without incredible supports from my advisor: David Lo, Associate Professor, Singapore Management University.

7. REFERENCES

[1] M. Asaduzzaman, C. K. Roy, S. Monir, and K. A. Schneider, "Exploring api method parameter recommendations," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*. IEEE, 2015, pp. 271–280.

[2] V. Bauer, L. Heinemann, and F. Deissenboeck, "A structured approach to assess third-party library usage," in *ICSM*, 2012, pp. 483–492.

[3] D. Bianchini, V. De Antonellis, and M. Melchiori, "Advanced web API search patterns adding collective knowledge to public repository facets," in *Proceedings of the International Conference on Information Integration and Web-based Applications & Services*, vol. 211, 2013, pp. 211–219.

[4] M. Bruch, M. Monperrus, and M. Mezini, "Learning from examples to improve code completion systems," in *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2009, pp. 213–222.

[5] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Improving IR-based traceability recovery via noun-based indexing of software artifacts." Wiley Online Library, 2012.

[6] W.-K. Chan, H. Cheng, and D. Lo, "Searching connected API subgraph via text phrases," in *Proceedings of the International Symposium on the Foundations of Software Engineering*, 2012, p. 10.

[7] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. T. Devanbu, "On the naturalness of software," in *Proceedings of the International Conference on Software Engineering*, 2012, pp. 837–847.

[8] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002, pp. 133–142.

[9] R. Lämmel, E. Pek, and J. Starek, "Large-scale, AST-based API-usage analysis of open-source Java projects," in *SAC*, 2011, pp. 1317–1324.

[10] F. Long, X. Wang, and Y. Cai, "API hyperlinking via structural overlap," in *Proceedings of the International Symposium of the Foundations of Software Engineering*, 2009, pp. 203–212.

[11] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: helping to navigate the API jungle," in *Proceedings of the ACM SIGPLAN conference on Programming Language Design and Implementation*, 2005, pp. 48–61.

[12] C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher, "Recommending source code for use in rapid software prototypes," in *Proceedings of the International Conference on Software Engineering*, 2012, pp. 848–858.

[13] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, and A. Marcus, "How can i use this method?" in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 880–890.

[14] R. Robbes and M. Lanza, "Improving code completion with program history," *Automated Software Engineering*, vol. 17, no. 2, pp. 181–212, 2010.

[15] M. P. Robillard, "Automatic generation of suggestions for program investigation," in *Proceedings of the International Symposium of the Foundations of Software Engineering*, 2005, pp. 11–20.

[16] M. P. Robillard and R. DeLine, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[17] Z. M. Saul, V. Filkov, P. T. Devanbu, and C. Bird, "Recommending random walks," in *Proceedings of the International Symposium of the Foundations of Software Engineering*, 2007, pp. 15–24.

[18] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *Proceedings of the Working Conference on Mining Software Repositories*, 2013, pp. 2–11.

[19] C. Teyton, J.-R. Falleri, and X. Blanc, "Mining library migration graphs," in *Proceedings of the Working Conference on Reverse Engineering*, 2012, pp. 289–298.

[20] S. Thummalapenta and T. Xie, "Parseweb: a programmer assistant for reusing open source code on the web," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 2007, pp. 204–213.

[21] F. Thung, S. Wang, D. Lo, and J. Lawall, "Automatic recommendation of API methods from feature requests," in *ASE*, 2013.

[22] F. Thung, L. David, and J. Lawall, "Automated library recommendation," in *Proceedings of 20th Working Conference on Reverse Engineering (WCRE 2013)*, 2013, pp. 182–191.

[23] C. Zhang, J. Yang, Y. Zhang, J. Fan, X. Zhang, J. Zhao, and P. Ou, "Automatic parameter recommendation for practical api usage," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 826–836.

[24] Q. Zhang, W. Zheng, and M. R. Lyu, "Flow-augmented call graph: A new foundation for taming API complexity," in *Proceedings of the International Conference on Fundamental Approaches to Software Engineering*, 2011, pp. 386–400.

[25] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes." in *ACM/IEEE International Conference on Software Engineering*, 2004, pp. 563–572.