Source Code based On-demand Class Documentation Generation

Mingwei Liu*†, Xin Peng*†, Xiujie Meng*†, Huanjun Xu*†, Shuangshuang Xing*†, Xin Wang*†, Yang Liu*†, Gang Lv*†
*School of Computer Science, Fudan University, Shanghai, China
†Shanghai Key Laboratory of Data Science, Fudan University, China
{19110240019, pengxin, 20212010114, 20212010075, 18212010042, 18212010029, 18212010066, 19212010053}@fudan.edu.com

***Abstract*—In this paper, we present OpenAPIDocGen2, a tool that generates on-demand class documentation based source code and documentation analysis. For a given class, OpenAPIDocGen2 generates a combined documentation for it, which includes functionality descriptions, directives, domain concepts, usage examples, class/method roles, key methods, relevant classes/methods, characteristics and concepts classification, and usage scenarios.**

***Index Terms*—Documentation Generation, Knowledge Extraction, Information Seeking**

I. Introduction

Many projects lack updated developer documentations that can answer different questions about a given class. The answers to these questions can help the developers to understand the class and complete relevant maintenance tasks. It is thus desirable that a documentation for a given class can be generated in an on-demand way based on its source code and existing documentations.

In this paper, we present OpenAPIDocGen2, a tool that generates on-demand class documentation based source code and documentation analysis. It is an improved version of OpenAPIDocGen from our previous work [1]. Given a class, OpenAPIDocGen2 automatically generates the following descriptions for it by analyzing the source code and relevant development documentations (e.g., user guide).

1) **Functionality Description**. The functionality description of the class and its methods.

2) **Directive**. Directive sentences that describe the constraints or guidelines for the usage of the class and its methods.

3) **Domain Concepts**. Domain specific concepts that are relevant to the target software project.

4) **Usage Examples**. Code fragments that use the class and its methods.

5) **Class/Method Role**. The roles that the class and its methods play in the project.

6) **Key Methods**. The methods that implement the important parts of the class.

7) **Relevant Classes/Methods**. Other classes/methods that are relevant to the class and its method.

8) **Characteristics and Concepts Classification**. The characteristics of the class and its methods (*e.g.,* readable, writable, serializable) and related concept classification (*e.g.,* org.jabref.model.entry.BibEntry is a BibTex/BibLaTeX entry).

9) **Usage Scenario**. The scenarios where the classs and its methods can be used.

II. Proposed Solution

To generate different parts of the on-demand documentation, we consider different aspects of the source code and use different code and text analysis techniques such as static analysis, text classification, text generation, graph mining, and clustering. Deep learning techniques are also used as required to boost the performance of specific tasks. For the identification of domain concepts, software documentations such as user manuals are also considered.

1) **Functionality Description**. Functionality description is provided for a class and each of its methods. Method functionalities are usually specified by method comments. For a method that have no comments, we use a code summarization technique to generate a summary for it. Following the current trend of code summarization, we treat the task as a neural machine translation problem and train a deep learning model to generate a summary for a given method. To this end, we collect a large corpus of method-comment pairs from thousands of Java projects in GitHub and train a Seq2Seq model by using the corpus as the training data. The model takes a sequence of code tokens as the input (*i.e.,* the source code of the method) and generates a sequence of text words as the output (*i.e.,* the summary of the method). For a method that have comments, we train a text classifier to identify functionality descriptions from its comments. We use the text classifier for implemented in our previous work [2], which classifies a given sentence into three categories, *i.e.,* functionality, directive, and other. We split the comment into sentences and use the classifier to determine whether each sentence is a functionality description or not. For class functionality description we use the same text classifier to identify functionality descriptions from the class comments.

2) **Directive**. We extract directives for a class from both its comments and source code. To extract directives from comments, we use the same text classifier [2] to identify directive sentences from the comments of the class and its methods. To extract directives from source code, we follow the approach proposed in [3] to extract four most common parameter constraints (*i.e.,* Nullness, Nullable, Range Limitation,

Type Restriction) for each method of the class by analyzing condition checking statements.

3) **Domain Concepts**. We identify domain concepts from the source code and documentation relevant to the project (*e.g.,* user manuals) by combining a learning-based approach and an unsupervised approach. The learning-based approach is from our previous work [4], which automatically construct a domain glossary from source code and software documentation. The unsupervised approach only uses the documentation as corpus to identify domain concepts. First, the corpus is cleaned by removing special characters and splitting sentences. Then all N-gram (less than 5) phrases are extracted as candidate domain concepts and low-frequency phrases are filtered. After that we calculate the four statistical features that measure the quality of domain concepts, *i.e.,* TF-IDF (Term Frequency–Inverse Document Frequency), PMI (Point-wise Mutual Information), information entropy of left adjacent word and information entropy of right adjacent word. We combine them together and use thresholds to select high-quality domain concepts from the candidates. The domain concepts extracted by the above two approaches are merged together. For a given class relevant domain concepts are selected based on their similarity to the source code (including comments) of the class.

4) **Usage Examples**. We first identify the code fragments in the current project that involve the methods of the class. Then we group the identified code fragments into different clusters based on the text similarity and structure similarity of the code. After that, we select a representative code fragment from each cluster as a usage example of the class.

5) **Class/Method Role**. For the class and its methods, we use the following heuristic rules to identify their roles. First, we identify the role of each method (*i.e.,* Accessor, Mutator, Creational, Constructor, or Others) based on the name, the declaration, and the implementation details of the method. For example, the methods that simply return the values of class fields or check the boolean conditions of the values are classified as Accessor methods. Based on the results of method role identification, we further identify the role of each class, *i.e.,* Entity, Factory, Util, Pool, or Others. For example, The classes whose number of Accessor methods and Mutator methods accounts for more than 80% of the total number of methods are classified as Entity classes.

6) **Key Methods**. We use PageRank algorithm [5] to identify key methods in a class. We run the PageRank algorithm on the call graph of the project and calculate the scores for each method. Then the top ranked methods of the class are treated as the key methods.

7) **Relevant Classes/Methods**. We use SimRank algorithm [6] to identity the classes/methods that are relevant to a given class/method. We run the algorithm on the call graph of the project and calculate the similarity between each pair of classes and each pair of methods. Then for a given class/method, we choose the classes/methods that can be reached in two hops in the call graph as the candidates of relevant classes/methods. The candidate classes/methods are ranked according to their SimRank scores with the given class/method and the top ranked ones are chosen as relevant classes/methods.

8) **Characteristic/Concepts Classification**. Our previous work [7] extracted the concept classification, functionality and characteristics of API elements from API reference documentation. Using the same technique, we extract characteristic specification and concepts classification for each class and method based on analyzing their names, code structure relationships and comments.

9) **Usage Scenario**. Given a class, we consider two typical usage scenarios: 1) how to get the instances of the class; and 2) where the class can be used. For the former, we identify the methods that return an instance of the class and use these usage examples as the first type of scenarios; we identify the methods that take an instance of the class as input and use these usage examples as the second type of scenarios.

III. Online Demonstration

We applied OpenAPIDocGen2 to the open source project, JabRef[1]. The data sources we used include the source code and comments[2] and user documentation[3]. An online demonstration of on-demand class documentation generation for JabRef is available at: http://106.14.239.166:8080/DocGen/index.html#/. It takes as input a fully qualified name of a class and returns the generated documentation for the given class. Our implementation for OpenAPIDocGen2 [8] and its front end for demonstration [9] are available on GitHub.

References

[1] X. Peng, Y. Zhao, M. Liu, F. Zhang, Y. Liu, X. Wang, and Z. Xing, "Automatic generation of API documentations for open-source projects," in *DySDoc@ICSME 2018*, pp. 7–8.

[2] M. Liu, X. Peng, A. Marcus, Z. Xing, W. Xie, S. Xing, and Y. Liu, "Generating query-specific class API summaries," in *ESEC/SIGSOFT 2019*, pp. 120–130.

[3] Y. Zhou, R. Gu, T. Chen, Z. Huang, S. Panichella, and H. C. Gall, "Analyzing apis documentation and code to detect directive defects," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017*, pp. 27–37.

[4] C. Wang, X. Peng, M. Liu, Z. Xing, X. Bai, B. Xie, and T. Wang, "A learning-based approach for automatic construction of domain glossary from source code and documentation," in *ESEC/SIGSOFT 2019*, pp. 97–108.

[5] A. N. Langville and C. D. Meyer, "Deeper inside pagerank," *Internet Mathematics*, vol. 1, no. 3, pp. 335–380, 2004.

[6] G. Jeh and J. Widom, "Simrank: a measure of structural-context similarity," in *KDD*, 2002, pp. 538–543.

[7] Y. Liu, M. Liu, X. Peng, C. Treude, Z. Xing, and X. Zhang, "Generating concept based api element comparison using a knowledge graph," in *ASE 2020*.

[8] "Openapidocgen2," Aug. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3984658

[9] "Openapidocgen2 frontend," Aug. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3984750

[1] https://github.com/JabRef/jabref

[2] https://github.com/JabRef/jabref/releases/tag/v5.0-alpha

[3] docs.jabref.org