# Automatic Generation of API Documentations for Open-Source Projects

**Conference Paper** · September 2018

DOI: 10.1109/DySDoc3.2018.00010

CITATIONS
4

READS
75

**7 authors**, including:

Xin Peng
Fudan University
**119** PUBLICATIONS **1,238** CITATIONS
SEE PROFILE

Mingwei Liu
Fudan University
**10** PUBLICATIONS **64** CITATIONS
SEE PROFILE

Yang Liu
Nanyang Technological University
**396** PUBLICATIONS **5,441** CITATIONS
SEE PROFILE

Zhenchang Xing
Nanyang Technological University
**156** PUBLICATIONS **3,248** CITATIONS
SEE PROFILE

# Automatic Generation of API Documentations for Open-Source Projects

Xin Peng*†, Yifan Zhao*†, Mingwei Liu*†, Fengyi Zhang*†, Yang Liu*†, Xin Wang*†, Zhenchang Xing‡
*School of Computer Science, Fudan University, Shanghai, China
†Shanghai Key Laboratory of Data Science, Fudan University, China
‡Research School of Computer Science, Australian National University, Australia
{pengxin, 17212010079, 17212010022, 18110240048, 18212010066, 18212010029}@fudan.edu.cn, zhenchang.xing@anu.edu.au

***Abstract*—Open-source projects often have only incomplete and insufficient API documentations. To improve the efficiency of development and ensure the correctness of API usage, it is desired that the developers can be supported with automatically generated documentation based on a combination of knowledge from different sources. In this paper, we describe OpenAPIDocGen, a system that can automatically generate API Documentations for open-source projects, including an overview of the system and the data sources and techniques used to generate different parts of the documentation.**

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

## I. Introduction

Developers often use open-source projects in their software development tasks by programming with the Application Programming Interfaces (APIs) provided by the projects. An example for this is the Java API for Microsoft documents provided by Apache POI [1]. Unlike standard programming libraries or frameworks (e.g., JDK, Android), this kind of open-source projects often have only incomplete and insufficient API documentations. Therefore, developers often need to seek the required information from Q&A forums (e.g., StackOverflow) and other websites, or learn the required knowledge from the repositories (e.g., source code, issue tracking systems, mailing lists) of the projects. To improve the efficiency of development and ensure the correctness of API usage, it is desired that the developers can be supported with the so-called on-demand developer documentation [2], i.e., automatically generated high-quality documentation based on a combination of knowledge from different sources [3], [4], [6], [7].

In this paper, we describe OpenAPIDocGen, a system that can automatically generate API Documentations for open-source projects. The system combines the information from different sources, including source code, code comments, issues, commits, StackOverflow (SO) posts. It includes an API knowledge graph as the shared knowledge base and uses a series of techniques such as entity linking, machine learning, topic mining, static analysis to generate different parts of the documentation.

## II. Overview

When developers are using or considering to use APIs, they may have the following information needs.

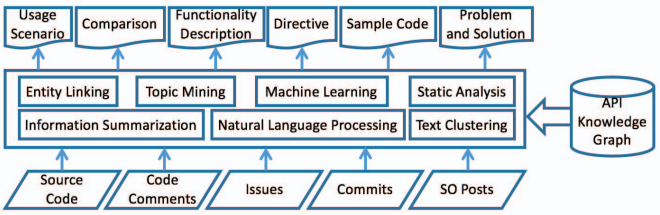


Fig. 1. Overview of API Documentation Generation

1) **Usage Scenario**. The usage scenarios where an API class can be used and the roles that the class plays;

2) **Comparison**. The comparisons of relevant APIs classes that can be used for similar purposes;

3) **Functionality Description**. The functionality descriptions of an API class, its methods and their parameters, and its attributes;

4) **Directive**. The constraints and guidelines for the usages of an API class and its methods;

5) **Sample Code**. The sample code about how to use an API class and its methods;

6) **Problem and Solution**. The problems that may be encountered when using an API class and its methods and the suggested solutions for the problems, or the known limitations of the API class.

An overview of the API documentation generation is shown in Figure 1. It extracts and combines information from different sources, including source code, code comments, issues, commits, and SO (StackOverflow) posts. An API knowledge graph is constructed to facilitate the generation of API documentation. It describes the relationships between API elements (e.g., classes, methods, parameters) and the relationships between API elements with background knowledge concepts (e.g., sheet, row, cell for Excel files). The API knowledge graph provides the shared knowledge base for the understanding of the information obtained from different data sources and the generation of different parts of the API documentation. A series of techniques are used to extract and analyze relevant and useful information from different sources and synthesize the required information for API documentation. For example, entity linking is commonly used in the generation of different parts of the documentation to relate text descriptions to the

978-1-5386-7570-0/18/$31.00 ©2018 IEEE
DOI 10.1109/DySDoc3.2018.00010

involved API elements. Also, machine learning (including deep learning) is commonly used to select useful information for different parts of the documentation.

## III. Proposed Solution

The construction of the API knowledge graph includes two phases. In the first phase, a skeleton knowledge graph is constructed by extracting API elements and their relationships from the source code of the open-source project. In the second phase, background knowledge concepts are selected from existing knowledge bases such as WikiData [5] and linked with API elements based on their descriptions and contexts. Based on the API knowledge graph, entity linking identifies mentions of API elements and background concepts in text and links the mentions to corresponding entities in the API knowledge graph.

The different parts of the API documentation for an API class thus can be generated in the following way.

**Usage Scenario**. The usage scenarios of an API class are identified in three steps. First, we identify discussions about the API class in SO posts and issues using entity linking. Second, we train a classifier that recognize descriptions about usage scenarios from the identified discussions. Third, we group the recognized usage scenarios into different clusters to identify representative scenarios. To explain the role of the API class in a scenario, we identify all the API classes that are involved in the same scenario and analyze their relationships based on the API knowledge graph. We then generate a graph and related descriptions that explain the role of the API class.

**Comparison**. The generation of API comparisons includes two parts: identification of potentially relevant API classes and summarization of discussions about these APIs. For the first part, we collect all the text about an API class from different sources (e.g., SO posts, code comments, issue descriptions) and build a document for it. Then we generate a document vector for each API class using word embedding techniques and generate a graph vector for each API class based on the API graph. The identification of potentially relevant API classes thus can be done by calculating a combined similarity between API classes based on both their document vectors and graph vectors. For the second part, we identify all the discussions from SO posts and issues that mention both the API class and a potentially relevant API class and use topic mining to summarize the main viewpoints about their comparisons.

**Functionality Description**. The functionality descriptions of API elements are generated in three steps. First, we collect all the text descriptions about an API element from different sources (e.g., SO posts, issues) using entity linking. Second, we identify a set of meaningful keywords for the API element by analyzing its text descriptions from different sources. Third, we train a seq2seq neural network based on existing functionality descriptions of API elements. The network takes as input the source code of the API element and a set of keywords and produces a sentence describing the functionality of the API element.

**Directive**. The directives of an API class are extracted from two different sources. One source is the source code of the class, from which method invocation constraints are extracted by analyzing the parameter checking and exception handling behaviours of related methods. The other source is the suggestions about the usage of the class from SO posts and issues. To extract directives from these suggestions, we first find relevant suggestions using entity linking, and then identify sentences that state meaningful directives by training a classifier. The identified sentences are then grouped into directives about the class or its methods.

**Sample Code**. The generation of API sample code includes three steps. First, we identify relevant code fragments and their surrounding text descriptions that involve the target API class. Second, we group the identified code fragments into different clusters based on their code structures and text descriptions. Third, we choose a representative code fragment from each cluster as a sample code and generates a description for it by summarizing the text descriptions of the code fragments in the cluster.

**Problem and Solution**. The generation of problems and solutions for an API class includes three steps. First, we identify questions and accepted answers about the API class using entity linking. Second, we select meaningful sentences about API usage problems and solutions by training a classifier. Third, we group together similar API problems and solutions and generate a problem/solution description for each group.

## IV. Online Demonstration

An online demonstration of OpenAPIDocGen is available at: http://bigcode.fudan.edu.cn/OpenAPIDocGen.


## Acknowledgment

This work was supported by the National Key Research and Development Program of China under Grant No. 2016YFB1000801.


## References


[1] Apache POI, http://poi.apache.org/index.html, 2018.

[2] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. A. Ernst, M. A. Gerosa, M. W. Godfrey, M. Lanza, M. L. Va squez, G. C. Murphy, L. Moreno, D. C. Shepherd, and E. Wong, On-demand developer documentation, in 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017.

[3] H. Li, S. Li, J. Sun, Z. Xing, X. Peng, M. Liu and X. Zhao, Improving API Caveats Accessibility by Mining API Caveats Knowledge Graph, in 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018.

[4] B. Xu, Z. Xing, X. Xia, and D. Lo, Answerbot: automated generation of answer summary to developersz technical questions, in Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017.

[5] Wikidata, https://www.wikidata.org, 2018.

[6] Y. Huang, C. Chen, Z. Xing. T. Lin and Y. Liu, Tell Them Apart: Distilling Technology Differences from Crowd-Scale Comparison Discussions, in Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering, ASE 2018.

[7] Q. Huang, X. Xia, Z. Xing, D. Lo and X. Wang, API Method Recommendation without Worrying About the Task-API Knowledge Gap, in Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering, ASE 2018.