CLEI electronic journal, Volume 24, Number 2, Paper 5, July 2021

Scout-bot: Leveraging API Community Knowledge for Exploration and Discovery of API Learning Resources

George Ajam
College of Information Technology, University of Babylon
Hilla, Babylon, Iraq.
george@itnet.uobabylon.edu.iq

Carlos Rodríguez
Departamento de Electrónica e Informática,
Universidad Católica Nuestra Señora de la Asunción
Asunción, Paraguay.
Visiting Fellow, School of Computer Science and Engineering,
UNSW Sydney, NSW, Australia
carlos.rodriguez@uc.edu.py

Boualem Benatallah
School of Computer Science and Engineering,
UNSW Sydney, NSW, Australia.
b.benatallah@unsw.edu.au

Abstract

Application Programming Interface (API) is a core technology that facilitates developers' productivity by enabling the reuse of software components. Understanding APIs and gaining knowledge about their usage are therefore fundamental needs for developers. Here, API documentation plays a pivotal role in enabling developers to take full advantage of the benefits brought by APIs. The quality of API documentation has therefore become an important concern given the celerity and dynamics at which APIs are now being made available to users. This article aims at exploring existing research in the area of API documentation in order to identify the associated quality dimensions addressed by the literature. The research is carried out as a Systematic Mapping Study (SMS) where 103 research papers selected from the literature were reviewed and a total of 5 core quality dimensions were identified and analyzed. By focusing on the two most relevant quality dimensions (understandability and completeness), this article presents an approach to enable API users to explore, discover and learn about APIs through API topic issues discussed in Stack Overflow (SO). We demonstrate the feasibility of our approach through *Scout-bot*, our tool for exploration and discovery of API topic issues.

Keywords: Quality of API documentation, Systematic Mapping Study, API topic issues, API Indexing, API Exploration

1 Introduction

Developers rely on several APIs for their day-to-day software development tasks [1] in a wide variety of scenarios, including cognitive services [2], Internet-Of-Things (IoT) services [3], data services [4], mashups [5], service-based business processes [6] and more. They also resort to several resources that document APIs. Examples of such resources include Community Question-Answering (CQAs) such as SO[1], API descriptions [7], code examples [8], among other resources. Yet, the problems of searching and finding API-related knowledge that satisfies the needs of developers are still open and they are far from being completely

[1] https://stackoverflow.com

solved. In this context, there are several studies in the literature on the analysis [9], extraction [10] and recommendation [8] of API-related knowledge such as usage examples and reference documentation.

One such line of studies includes the improvement of API documentation and learning resources. Works along this dimension focus on the problems of lack of API usage examples [11], missing descriptions in API documentation [12] and insufficiency of usage patterns [13]. Other studies include the factors affecting the usability of APIs [14] and code snippets in software-related documentation (including APIs) [7]. Further topics of interest in the literature include the production [15, 16, 17, 18, 19], problems/issues [20, 21, 22, 23] and enrichment [12, 13, 24, 25, 26] of API learning resources.

Despite all the studies and proposed approaches in this context (we elaborate more on this in Section 2), it is still challenging to explore and navigate API knowledge and their related issues through the various forms of API documentation [27]. This is partly due to the heterogeneous and fragmented information found across different sources, making it difficult for querying, exploration and understanding purposes. In this article, we extend our previous work [28] and present a systematic study of the literature that led us to identify key quality dimensions of API documentation. More specifically, we explored the literature through an SMS [29] that helped us identify such key dimensions. Armed with and motivated by the findings obtained from this SMS, we outline our proposal consisting in organizing API community knowledge into API topics issues to facilitate discovery and understanding. We focused on API topic issues, as opposed to traditional, plain keyword-based search, because it is more appropriate for discovery and learning purposes, particularly, in unfamiliar spaces [30]. In concrete, in this article we present:

- a systematic mapping study [29] that allows us to identify key quality dimensions of API documentation and their coverage by existing literature;
- an API-topic-centric data model for building the foundation for a Knowledge Base (KB) to represent and store relevant API knowledge and API topic issues;
- an API knowledge indexing technique to support API topic issues exploration, which enables the various API resources to be accessed conveniently in an API-topic-driven manner;
- an enrichment technique for API topics terminology that leverages word embeddings [31], which helps provide developers with more flexibility for expressing their API topic issues queries;
- *Scout-bot*: A query bot that helps querying API topic issues through a simple yet powerful domain-specific query language. We showcase how such query bot can be implemented and seamlessly integrated into existing productivity tools.

We organize this paper as follows: Section 2 presents the SMS on quality dimension of API documentation. Section 3 introduces our model and approach to indexing and enrichment of API topic issues through API community knowledge. Section 4 presents our Domain-Specific Language (DSL) for querying API topic issues. In this same section, we also propose *Scout-bot*, a bot for querying the API topic issues enriched index, including its implementation details and evaluation. We conclude this article with Section 5 presenting the final discussions and future directions for this research work.

2 Quality Dimensions of API Documentation

APIs have become the cornerstone of business and software integration, and a key enabler for technologies such as data services [4], robotic process automation (RPA) [32], blockchain [33] and IoT [3]. Developers typically learn about APIs through *API documentation*, which is considered a valuable resource that aims at providing, among other things, descriptions and specifications of programming interfaces, API usage knowledge and examples, and guidelines and best practices [34, 35, 36, 37]. As such, API documentation is widely used both as reference for developers while performing their development tasks and as a learning resource [34, 35].

The Web has substantially changed the way APIs are documented [20]. Q&A websites [14], social media [20] and even video tutorials [38] are nowadays widely used to document both software in general and APIs in particular [28, 39]. Regardless of its significant role in the software development process, API documentation face several quality issues that can drive developers off an API and force them to use a different one [34]. As a response to this problem, researchers have started to explore the issue to provide more insights and better understanding of the problems [20, 34, 40, 41, 42, 43, 44, 45, 46]. For example, Uddin and Robillard [20] explored the issues that industry developers face regarding two main areas, namely, the content of API documentation and the way it is presented to developers. In this section, we focus on exploring the quality dimensions of API documentation addressed in the literature. More specifically, we aim at addressing the following two research questions:

Table 1: Combination of search terms used in our SMS study

Primary terms set	Secondary terms set
"API Documentation"	"Quality"
"Application Programming Interface Documentation"	
"Crowdsourced API Documentation"	

- ***RQ1:*** *What quality dimensions are being addressed in the literature in the context of API documentation?*
- ***RQ2:*** *How is current research mapped to these quality dimensions and to what extent are they being covered?*

In order to address each of the research questions above, we propose to explore the existing literature using an SMS [29]. SMS is a research method that helps in characterizing a research area and computing the extent of the contributions in the literature under each identified category. The main goal of the method is therefore to provide structuring for an area of research. We examined a total of 103 research papers reporting on studies related to API documentation and its quality, and identified a total of 5 key quality dimensions for API documentation, which we explored using thematic analysis [47]. Based on the identified quality dimensions, we report on the number of publications under each quality dimension, characterize existing research in terms of the types of venues they were published in, report on the research methods used, and discuss the main trends that emerge from the selected papers.

2.1 Research Method

We use a systematic mapping study to address the research questions above. The main goal of SMSs is to categorize a research area of a topic in the existing literature [29]. Other objectives include the definition of an unknown research area and the summarization of findings. We followed the updated guidelines for conducting a SMS provided by Peterson et. al [29]. Similar to Systematic Literature Reviews (SLRs) [48], SMSs also look at surveying the literature using a comprehensive and rigorous methodology, except that an SMS looks at higher-level research questions. Thus, the SMS aims at creating a structure for a given research field and focuses on the overall trends [29]. We also followed the steps for thematic analysis [47] to explore the quality dimension themes. The thematic analysis steps are similar to those of SMS except that it helps reduce overlaps by creating a model of higher-ordered themes, typically 5 to 7 themes as recommended by Cruzes and Dyba [47].

2.2 Search Strategy

Our literature search expands from 2009 to 2020. It was carried out mainly on three databases: IEEE Xplore Digital Library[2], ACM Digital Library[3] and Google Scholar[4]. We chose the former two digital libraries because of their focus on computing literature. The latter is used to complement and improve coverage of our search [49, 50] (we considered only the first 100 results, because advancing beyond this did not produce relevant results for our search). We queried each of these databases with combinations of both the primary and secondary terms as shown in Table 1. We show examples of such combinations in Table 2. Search terms we used varied according to the search capabilities of the publication database. For instance, IEEE Xplore Digital Library has an advanced search option that allows users to add a specific keyword to be searched in the title, abstract and other metadata of the publication, which, when combined with Boolean expressions, can make the search more effective. In addition to the search mechanisms discussed previously, we also carried out a manual search for key conferences (e.g., The International Conference on Software Engineering (ICSE)) and journals (e.g., IEEE Transaction on Software Engineering (TSE)) in the area of Software Engineering. Backward snowballing [51] was also applied in order to find studies related to API documentation that were missed through the search strategy above.

2.3 Selection Criteria

The inclusion criteria for selecting a study is based on the following: (1) The study addresses API documentation production, improvement and problems, and it generally aims at improving software development

[2]https://ieeexplore.ieee.org
[3]https://dl.acm.org
[4]https://scholar.google.com

Table 2: Examples of queries used in our SMS

Database	Search Terms example
IEEE Xplore Digital Library	((API documentation) AND "Abstract":quality) ((API documentation) AND "Document Title":Quality) (("Abstract":API documentation) AND "Document Title":Quality)
ACM Digital Library	(+API +Documentation +Quality)
Google Scholar	intitle: API documentation quality "API Documentation" AND "quality"

Table 3: Number of included studies from the selected sources

Database	Included Studies
IEEE Xplore Digital Library	28
ACM Digital Library	17
Google Scholar	48
Conferences & Manual Search	10
Total	103

tasks by leveraging API documentation resources; (2) the study aims at improving user experience when using APIs and the corresponding documentation. The exclusion criteria for any given study is as follows: (3) The study is not written in English; (4) the study was not peer-reviewed; (5) the study is not accessible in full-text; (6) the study is a duplicate of another study; (7) the study is about mining code, patterns of usage or code recommendation for purposes other than the improvement of API documentation.

A total of 4,006 papers was obtained as the result of the search. By applying our selection criteria, we ended up with a filtered list of results consisting of 103 research papers, where a big majority of the filtered-out papers were out of scope (e.g., medical studies and API product documentations). Table 3 shows the results obtained from each of the sources we used for our search.

We can see that the majority of the papers were obtained through Google Scholar search, accounting for a total of 48 unique papers (∼47%). IEEE Xplore, instead, provided a total of 28 papers (∼27%), while ACM Digital Library contributed with 17 papers (∼17%). The additional manual search provided a total of 10 papers (∼10%) that were not found through the search in the previous sources.

The study types of the research papers, based on the types discussed in Petersen et al. [29], are shown in Fig 1(a). We can see that a majority of the studies are dedicated to *solution with validation* (∼59%), followed by *evaluation research* (∼28%), *philosophical research* (∼8%) and *proposed solution* (∼5%). The number of papers in the latter two categories is small in comparison with the previous two. Fig 1(b), instead, focuses on the research methods used in the studies. We can observe that *design science* represents ∼57% of the research methods used, followed by ∼16% *empirical studies* and ∼14% *case studies*. Here, we can notice that less research work is conducted using *mix methods* (∼8%), *qualitative studies* (∼3%), *quantitative studies* (∼2%) and *field studies* (∼1%). These results also indicate that the main trend lies in research that focuses on designing solutions (*design science*) (see Fig. 1(b)) that are then validated and evaluated (see Fig. 1(a)). Regarding the types of venues in which the papers were published (see Fig. 2), ∼79% of the papers were published in *conference proceedings*, ∼14% in *journals*, ∼7% in *symposiums* and ∼1% as a *technical report*.

2.4 Results

2.4.1 Quality dimensions for API documentation

We first focus on addressing ***RQ1*** - *What quality dimensions are being addressed in the literature in the context of API documentation?* In order to answer this question, we followed the steps of thematic analysis [47] to analyze each of the 103 papers obtained from our search. More specifically, the thematic analysis steps followed were [52]: (i) Initial reading of the text (papers), (ii) identification of specific segments of text, (iii) coding of the segment of text, (iv) reduction of overlaps in coding and translation of codes into themes, and (v) creation of higher-order themes. The results of this analysis helped us identify five quality dimensions addressed in literature, which we summarize in Fig. 3. The ellipses represent the identified quality dimension, while the rectangles indicate the main artifacts to which they are applied in the selected papers.

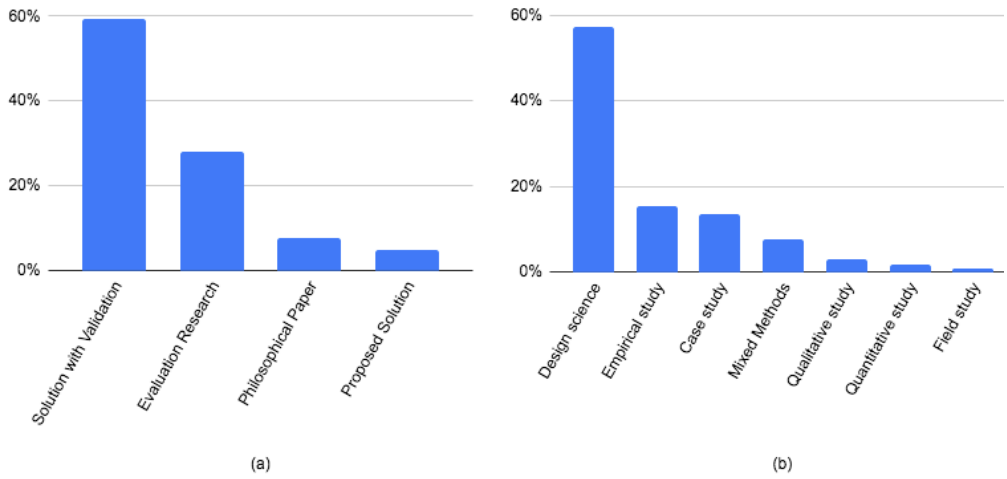

Figure 1: Study types and research methods used in the selected papers

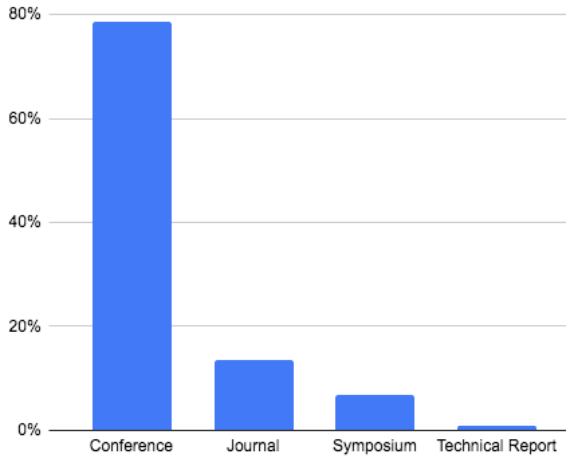

Figure 2: Venue types

Completeness. In literature, the term documentation completeness is a desired quality generally required to mitigate problems of missing resources and descriptions [40]. The inclusion of all descriptions of API elements in the documentation is necessary so that readers can find information about how to properly use an API. Thus, an API documentation is incomplete when it misses important information, conditions and knowledge elements that is required by API users. Several works from the literature focus on the issue of completeness [10, 25, 37, 44, 46, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91]. In our findings, the thematic analysis showed that the main artifacts to which completeness is applied are resources, directives and summaries. Resources represent in this context usage examples, tools, tutorials and examples of requests and responses to provide support for the content of developer and reference documentation. To improve completeness of available resources, studies investigated tooling and examples of API usage [25, 37, 53, 54, 55, 56, 60, 63, 64, 65, 66, 67, 68, 69]. For example, to help API developers find examples related to specific APIs, Mar et al. [57] designed PropER-Doc, a system that takes queries from developers and uses code search engines to find related code examples and apply appropriateness metric to group these examples [57]. In the context of method invocation, the problem of not keeping the developers aware of the rules related to these methods may lead to errors while the developers write code. Directives, on the other hand, refer to clear contracts that help understand what is possible and what is not (i.e., constrains) while using an API method [63]. Dekel and Herbsleb [60] proposed a tool called eMoose that support the Eclipse IDE[5] for the decoration of API methods that have rules and directives, providing them as tool-tips and by highlighting the related directives from the original API documentation. Yet, if directives are missing from the documentation, the user still has to find alternative resources.

Thummalapenta and Xie [92] propose a tool named Alattin that uses mining techniques to detect rules

[5]https://www.eclipse.org/ide

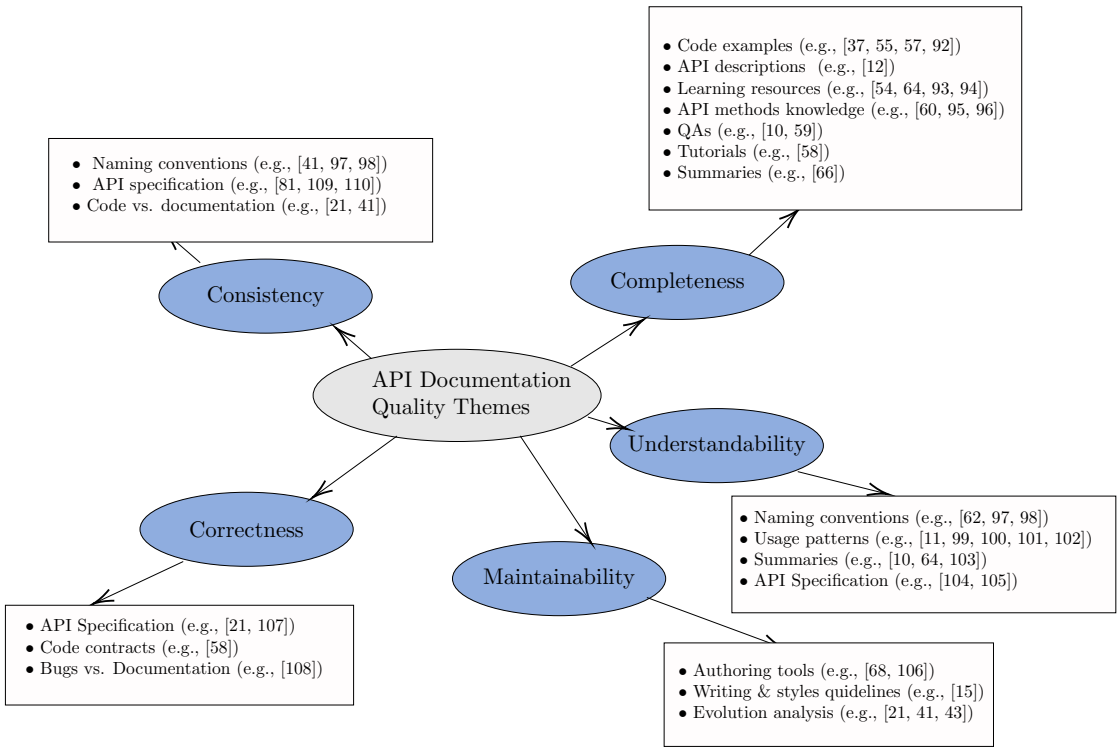

Figure 3: Thematic map for quality dimensions of API Documentation

and conditions that are neglected in documentation. While this method can be considered a detection mechanism of directive issues, it can also be applied to check for software defects, code and patterns violations, and as a support for quality control for documentation completeness [64]. Maleshkova et. al [65] examined Web APIs to analyze APIs forms and descriptions. The authors reported on completeness of APIs documentation, showing that more than 75% of APIs has usage examples, while 53.1% include descriptions of error messages. These results are based on manual analysis of ProgrammableWeb[6] (a Web API directory), where details are usually added manually. The authors questioned the accuracy of the information added on ProgrammableWeb, highlighting errors such in feature descriptions, URL links, and authentication information.

In a different direction, Hou and Li [66] performed an exploratory study on the forum discussions about API learning obstacles. API usage problems identified in [66] include obstacles related to "Asking for a solution", "Using a wrong solution" and "Using a solution incorrectly". Incompleteness of API documentation and undocumented API affected mostly the first category (i.e., "Asking for a solution"). This indicates that such problems stem from APIs that do not have documentation, thus causing difficulties in carrying out the steps required to effectively use the API. Similarly, Robillard and Deline [67] explored, through surveys and interviews, the classes of obstacles professionals face in terms of availability and quality of learning resources. Interestingly, the most severe obstacles found are inadequate documentation and code examples. Even when documentation was present, it was found that documentation with high-level abstractions is the most severe barrier that developers face, perhaps as a result of the need for clear descriptions and explanations.

Other aspects such as the structure of APIs, the environment and the background of developers have less effects [67]. In order to fill the gap of linking resources to API documentation, Degenais and Robillard [68] proposed a technique that looks at documentation of an open source project to link terms related to code with code elements. Parning et al. [69], instead, studied the crowd contributions to API documentation by looking at the discussions of Stack Overflow. They reported that contributions such as a Q&As discussions, achieved high coverage of API functionality along with usage examples. The contributions have been curated by the crowd for the purpose of updating the content and correcting error and mistakes. It is notable that crowd contributions may not cover all aspects of a specific API, but it may help with API documentation completeness by looking at descriptions, code-like terms linking, usage examples and even summaries. Chen and Zhang [10] designed a system that uses both visits of developers to API documentation and Q&As, integrating the corresponding Frequently-Asked-Questions (FAQs) from this system into the API documentation. Petrosyan and Robillard [58] propose an approach to help discover tutorial sections that are related

[6]https://www.programmableweb.com

to the explanation of a specific API types. In the same line, Treude and Robillard [59] also contributed to the explanation of API types. Their system uses machine learning techniques to automatically leverage on features from the questions and answers along with their similarity to the corresponding API documentation. The results are then embedded into the API documentation to assist developers understand the API type of interest.

Other studies looked at directives to explore usage conditions [60], contracts [61] and constrains issues [62]. For example, Gao et. al [61] inferred data contracts out of Web APIs by analyzing error messages and using decision tree learning method. While Saied et al. [62] proposed automatic methods to detect usage constraints. One of their findings is that constraints are specifically absent from the documentation of APIs. Finally, Watson et. al. [44] reviewed 33 API documentation from open source projects. Their survey provides insights into the design and writing quality used along with elements of a typical API documentation.

Understandability. This dimension is related to the clarity of API documents so that developers can understand the usage and purpose of an interface [40]. Several studies have addressed this dimension in the context of API documentation [8, 40, 46, 58, 59, 60, 70, 72, 74, 75, 76, 77, 78, 79, 80, 82, 83, 84, 85, 87, 88, 89, 91, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102]. Some of these studies have noted the importance of the need for enrichment, such as summaries of text around code in CQAs [96] and supporting documentation with insights [59], observing also that missing good examples affect users who, as a consequence, often look for alternative resources on the web [95]. Parnin and Treude [94] studied the coverage of API documentation in the web and found that the highest frequency are for tutorials related to APIs followed by experience blog posts, code snippet, among other types of documentation spread across Q&As like Stack Overflow, forums, official documentation and mailing lists. Petrosyan et al. [58] proposed a discovery technique for tutorials related to API types by using supervised text classification. This approach helps in making developers aware of related tutorial sections that can explain questions of the type "how-to" with good relevant explanations about the API type. Similarly, Buse and Weimer [93] propose to synthesize API code usage examples in a well-typed, readable and representative form with focus on the readability of the generated examples as if they are written by humans [93].

In a different direction, naming conventions were explored to inform an easy-to-read, consistent API document [98]. For example, sorted methods names can help authors spot inconsistencies and ambiguities in such names. Directives is also considered as a mechanism to enhance understandability of API documentation. Notably, studies [60, 99, 100] addressed such issue using techniques such as mining, providing knowledge about directives and empirically studying the effects of failures in identifying directives in code fragments with errors. Wu et al. [103] propose the abstraction of functionality in diagrams reflecting the semantics of API calls as a way to improve the understandability of APIs. Finally, Montandon et al. [8] uses static slicing algorithm to summarise code examples extracted from private code repositories to supplement API documentation.

Maintainability. Maintainability includes studies that are related to the process of correcting errors or conflicts found in API documentations (e.g., [21, 42, 43, 56, 61, 71, 72, 73, 74, 81, 83, 86, 104, 105, 106, 107]). The improvement of API documentation maintenance has been explored in two areas. The first one relates to documentation evolution. For example, the case study on Web APIs presented in [21] provide an understanding on how the content of API documentation changes during events such as versioning. These changes were observed in manual processes, which expose the maintainability of documentation to human errors. The second area is about understanding and implementing the tooling required to actually maintain evolved documentation. In [42], maintainability was considered as an attempt to understand contributions of developers in open source projects. In this study, Dagenais and Robillard studied 19 documentation revisions and also explored how developers participate, contribute and read documentation. The study reported that contributions in public wikis led to increased maintainance, lowered the entry barrier and led to low quality [42]. Because API documentation can become outdated, Subramanian et al. [56] proposed Baker, a tool that supplement API documentation with up-to-date code examples based on Q&As and Github Gists[7]. Detection of errors [43] about deprecated API methods and usage error messages are also taken into account for maintaining data contracts [61]. Expanding on this area can be an interesting research direction, especially as a systematic search of how maintainability is supporting API documentation evolution and correctness.

Consistency. Consistency measures are connected to documentation content and the way it is presented by means of no conflicts in the descriptions of artifacts and their presentations [40]. That is, an API documentation is consistent when information given about a specific element of an API is uniform across all content and versions of the API documentation. Consistency also means that presentation of content and format follow regular usage of style and visualization across all documentation. From our findings, we can observe an initial objective aimed at detecting and highlighting API documentation issues and

[7]https://gist.github.com

inconsistencies [14, 41, 71, 80, 81, 97, 101, 105, 106, 108, 109]. For instance, Correia et. al [109] highlighted the main consistency problems of documentation along with proposed solutions. Inconsistency in this case is a result of evolution of some parts of the documentation away from other parts. In order to keep API documentation consistent with actual functionalities or data, Zhong and Su [41] introduce a tool called DOCREF for detecting errors and inconsistencies in documentation. Another objective in the context of API documentation consistency is the usage of specific styles and guides for authors [15]. These include understanding the workflow, where incremental contributions in collaborative settings are encouraged to support documentation changes.

Correctness. This quality dimension focuses on the validity of API documentation knowledge elements, with several works addressing the issue [14, 20, 56, 66, 71, 74, 75, 76, 84, 101, 108, 110, 111, 112]. Incorrect API documentation has been reported [20, 66] to cause challenges to users while using and learning APIs. We have identified that incorrect API documentation are either conceptual, which are related to errors in the descriptions (e.g., natural language description of an API method), or code-related, such as in cases where bugs are present in code examples provided in API documentations. Errors in descriptions affect API semantics [56], while code related bugs affect the usability of APIs [14]. Moreover, Zibran et al. [14] found that 34.9% of the bugs result from incorrect API documentation, including incorrect descriptions of a specific API functionality. Along this line, Daielsen and Jeffrey [110] propose to support correct API descriptions using a machine-generated documentation. The so-generated documentation can be validated with formal representation by embedding a proposed annotation similar to RDF within the documentation's HTML page [110]. While Zhong et al. [111] propose a method that combines Natural Language Processing (NLP) techniques with code parsing in order to detect API documentation errors. The resulting approach (DOC2REF) can handle API documentation errors with a focus on grammatical errors among other features.

2.5 Quality Dimensions Coverage for API documentation

In this section, we focus on ***RQ2*** - *How is current research mapped to these quality dimensions and to what extent are they being covered?* In order to answer this question, we map the papers obtained from our search to the five quality dimension identified through our thematic analysis, and we associate them to the different research types of each study (see Fig 4).

Our findings show that most studies focus on *solutions with validation*. These include tools to support *understandability* (∼28%) through directives and methods, conventions and need for enrichment; while studies on *completeness* (∼28%) seem equally relevant to researchers making these the two most common quality dimensions covered by the literature. Next, an important number of works focus on *evaluation research*. Again here, *completeness* (∼20%) is a popular concern where research efforts are put toward understanding and unveiling problems of missing resources and descriptions and their mitigation mechanisms. A much lower percentage of the analyzed papers are about *proposed solutions* without evaluation and *philosophical research*.

2.6 Disscussion and Threat to Validity

The results of this study show a clear concentration of studies in the quality dimensions of *completeness* and *understandability*. This comes as no surprise given that developers expect that API documentations provide a full reference of the interfaces that not only assist them in their day-to-day development tasks related to API usage but also help them learn about the existing and new APIs [113]. This results in a response from the research community that tries to provide a more complete and holistic API documentation that empower developers through code examples (e.g., [103]), Q&As (e.g., [114]), tutorials (e.g., [38]), and other types of API documentation enrichments (e.g., [58, 115]). Fewer works focus on the remaining quality dimensions, which however does not make them less important. Works regarding maintainability focus mostly on API documentation evolution (e.g., [11, 116, 117, 118]), while, to a lower extent, focus is put on documentation errors (e.g., [119]) and creation (e.g., [120]). When it comes to consistency, researchers focus on patterns for helping in the creation of consistent documentation (e.g., [121]), making sure consistency of documentation is kept as APIs evolve (e.g., [116]) and the effects of documentation consistency on API usability (e.g., [122]). Finally, correctness of API documentation has triggered research in areas related to API specification (e.g., [111, 123]), usability (e.g., [124]) and automatic generation of usage examples (e.g., [123]).

Despite the considerable efforts by the research community, there is still much space for improving the quality of API documentation. As unveiled by Uddin and Robillard [20], there are still major concerns emerging from practitioners regarding the incorrectness, ambiguity and incompleteness of API documentation. The recent advances in Artificial Intelligence (AI), NLP and Natural Language Undersdanding (NLU) are a good fit to address these issues given the unstructured nature of large parts of API documentation.

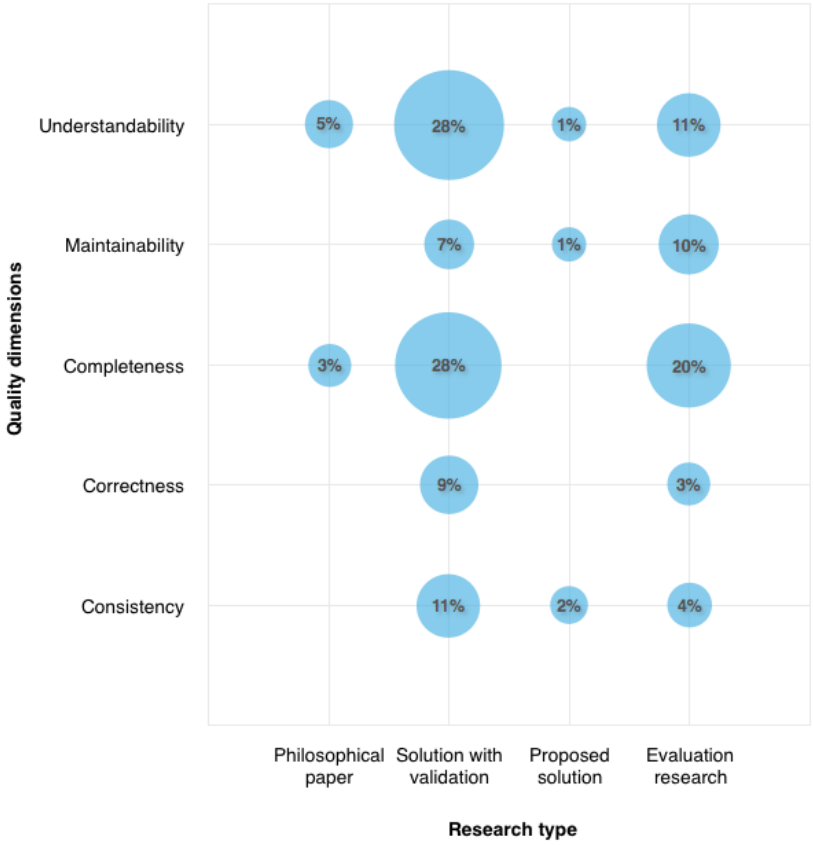

Figure 4: Quality dimensions and research types used in the selected papers

Initial attempts along these lines include the use of NLP for Web APIs documentation [125], API embeddings for API usage/application [126] and API document embeddings [127]. We argue that further research is needed to assess API documentation quality, e.g., by leveraging embedding technologies on top of both structured (API formal specification) and unstructured data (API documentation).

This study has its own *limitations*. The examination, selection process and the data extraction strategy were carried out by only one examiner, which may increase the risk of human error and bias. While making use of regular meetings with the authors to discuss the outcomes may have helped in mitigating this issue, this threat may still affect the results of this study. The selection of search databases was also limited to two of the most relevant digital libraries for Software Engineering. This limitation was partly mitigated by also using Google Scholar, which contributed to finding results beyond what was provided by these two databases [49, 50]. The set of query terms was also limited by the capabilities offered by the search engines of the two digital libraries and Google Scholar. In this case, our mitigation strategy consisted in using backward snowballing and manual inspection of conferences and journals that are relevant to the Software Engineering community.

3 Modeling, Curation and Indexing of API Community Knowledge

The SMS presented in the previous section shows that *completeness* and *understandability* are two of the most important quality dimensions for API documentation in terms of how much attention they have received by researchers. In this context, previous research highlights the common practice, by developers, of searching for alternative resources whenever they face completeness and understandability issues with API documentations (e.g., [69, 94]). A key challenge related to this search task is that of being able to query, explore and discover these alternative resources. In this and the following sections we present our approach to facilitate the exploration and discovery of API learning resources by leveraging API community knowledge.

Several resources can be found nowadays on the Web that can help mitigate existing API documentation issues. Among these resources, SO is one of the most widely used and indispensable resources for facilitating the usage and understanding of APIs [128]. It features a vast repository of programming knowledge including approximately 21 million questions, 32 million anwers, 15 million users and 11 million visitors per day, as

of April 2021.[8] Study [39] has shown that API-related posts in SO can be categorized into topic issues such as API security, API usage, API Debugging, among other topics issues. In this section, we present our data model to represent, enrich and index API-related posts in SO based on such topic issues, in order to support discovery and understanding of API-related knowledge. We discuss next the data model as well as the extraction/curation and indexing techniques we use to represent and index such resources.

3.1 Data Model

Given the relevance of SO in the programming community, we developed a SO- and API-topic-centric data model to capture knowledge about APIs. Leveraging on our experience and results from previous research [129], we introduce the model in Figure 5. Here, *API* is a software released in a current version represented by *API version*. An API version is typically accompanied by an *API documentation*, where the API documentation can be a reference documentation, getting-started guide or an open API specification (e.g., a Swagger API Documentation[9]). An *API version* contains several API methods represented by the *Method* entity of our model. *Insights*, in turn, are obtained from our information sources and include metrics about posts and developers using the API.

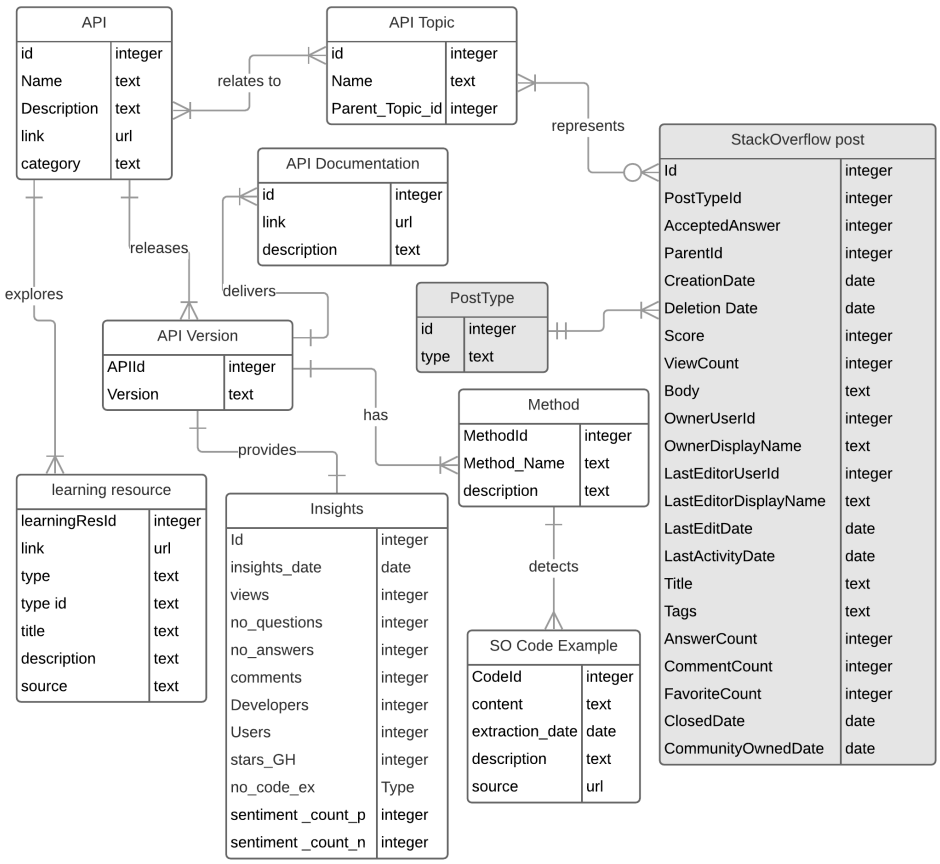

Figure 5: Data model used for the representation of API learning resources

Next, an *API Topic* represents a topic issue, as identified in [39]. *SO Code Example* represents code examples that contain a method from a specific API version. These code examples can be either obtained automatically or from a human-curated list of examples. An API, may contain several other learning resources such as getting-started videos from Youtube[10], code examples from Github[11], among other types of resources. Finally, and, most importantly, our data model includes *Stack Overflow posts*. This entity represents a question, answer or a Wiki[12] posted on SO.

[8] https://stackexchange.com/sites
[9] https://swagger.io
[10] https://youtube.com
[11] https://github.com
[12] https://meta.stackexchange.com/questions/11740/what-are-community-wiki-posts

Notice that, in addition to including entities and attributes from SO's dataset in our model (see Figure 5), a key characteristic lies in the API-topic-centric approach we followed in the model. Such modelling decision is key to building a solution that enables the indexing, exploration and discovery of API-related knowledge based on API topic issues as we will be discussing in the next sections.

3.2 API Community Knowledge Extraction and Curation

We leverage on the API community knowledge in SO and the data model introduced in the previous section in order to build our API KB. The first step toward extracting such API-related knowledge consists in identifying SO resources (e.g., posts) that relate to known APIs. Curated lists of known APIs can be obtained from existing directories of APIs such as ProgrammableWeb[13]. By leveraging SO's search APIs[14] we can construct keyword-based queries that can help us retrieve posts that relate to specific APIs. For instance, in order to search for posts related to Windows API[15] (or WinAPI for short), we can build a query containing the keyword "WinAPI" as tag, as shown in Figure 6. More complex queries can be built using SO's APIs, e.g., to include posts that contain specific keywords in the body of posts (see footnote 14 for more details).

```
1 GET /2.2/search?pagesize=100&order=desc&sort=activity&tagged=
    winapi&site=stackoverflow
```

Figure 6: Example of SO's search API to query for posts related to WinAPI

In SO, not all posts are properly tagged and curators from the community may add tags over time as needed. This can lead to missing the extraction of relevant content simply because posts were not tagged properly with the corresponding API name. We therefore resort to queries that retrieve posts based on APIs mentioned not only within tags but also title and body of posts. As explained before, such list of mentions can be obtained from existing list of APIs available on the Web (see footnote 13). Additionally, in order to assign topics to each post extracted from SO, we use a list API-topic-issues seed keywords, e.g., based on the topics listed in [39]. Such list of seed keywords are expanded with alternative mentions obtained from a pre-trained word embedding based on SO [130]. This way, whenever we identify mentions from our list of API-topic-issues in a post, we associate the post to the corresponding topic issue.

In addition to SO posts, we further curate our API KB by enriching it with additional resources such insights, code examples, API documentation, among other resources (see our model in Figure 5). Such enrichments can be sourced from other sites such as Github (e.g., for code examples) and Youtube (e.g., for video-tutorials). Github and Youtube also provide search APIs[16] [17] that can help retrieve API-related resources, which we leverage in our work.

3.3 Indexing

When an API consumer, programmer or practitioner is interested, e.g., in learning about "security" issues related to "Facebook Graph API", they typically need to make several assumptions and keyword-based guesses in order to find relevant information. This, in turn, may not bring all posts that tackle the target API issues, limiting search results only to exact keyword-based matches. While this type of search is popular in many platforms (e.g., SO) to look for information that help them solve issues related to specific programming tasks, it is not suitable for a topic-driven search of posts related to APIs. In this section, we report on how we leverage Elasticsearch[18] and the data model introduced before for indexing APIs on a topic-centric basis.

Elasticsearch is an open-source search engine that provides near real-time search services. It is built upon an open-source information retrieval and indexing library: Apache Lucene[19]. Elasticsearch provides REST APIs and supports the distributed aspects that Apache Lucene lacks. In Elasticsearch's terminology, an index is a container that stores JSON documents (e.g., posts in SO). The schema of an index is defined through so-called mappings[20]. Elasticsearch uses Apache Lucene to generate an inverted list [131] of keywords where each keyword is associated to documents that contain such keyword.

[13] https://www.programmableweb.com
[14] https://api.stackexchange.com/docs/search
[15] https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list
[16] https://developer.github.com/v3/search
[17] https://developers.google.com/youtube/v3/docs/search
[18] https://www.elastic.co
[19] http://lucene.apache.org
[20] https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html

```
{"mappings":
  {"_stackoverflow":
    {"properties":
      {
        "title":{"type":"text"},
        "body": {"type":"text"},
        "tags": {"type":"keyword"},
        "api": {"type":"keyword"},
        "topic": {"type":"keyword"},
        "question_id":{"type": "integer"}
      }
    }
  }
}
```

Figure 7: Elasticsearch mapping for indexing SO posts along with the *api* and *topic* mentions they relate to

We leverage on Elasticsearch indexing and searching capabilities to support exploration and discovery of API topic issues. In concrete, we index SO posts related to APIs and use the resulting index as a gateway to get access to our KB represented by the model in Figure 5. The index follows the mapping (i.e., schema) shown in Figure 7. In this mapping, the *title*, *body*, *tags* and *question_id* correspond to standard attributes from SO's dataset. Whereas the attributes *api* and *topic* are extended attributes that characterize each post based on the API (e.g., "WinAPI") and topic (e.g., "API Security") the post relates to.

APIs may be referred to using different mentions across different posts. For instance, "Windows API" can be referred to also as "WinAPI" and "Windows 32 API". The same rationale applies to API topic issues. In order to be able to account for different mentions of APIs and API topic issues, we propose to extend the values within the attributes *api* and *topic* (see Figure 7) to include different mentions thereof. We do such enrichment using word embeddings techniques [31], where words (from a given corpus) are mapped to a vector space. A key property here is that words that are semantically similar appear close to each other in the vector space. Several pre-trained models exist today [31, 130, 132, 133], including general-purpose embeddings (e.g., based on Wikipedia corpus) and domain-specific ones (e.g., programming corpora). We focus on the latter and leverage on a word embedding pre-trained model based on SO [130]. Furthermore, we leverage on previous work on building software engineering thesaurus [134] and API recognition in SO [135] in order to identify API-relevant terms from SO posts. Table 4 shows examples of SO posts enriched with additional mentions for APIs and API topic issues after the application of the aforementioned techniques.

The resulting index can thus be used to perform searches based on APIs and their associated topic issues. The corresponding queries can be written with terminology flexibility thanks to the enrichments discussed before (e.g., users can write queries related to Windows API by using the terms "WinAPI", "Windows API", etc). In the next section, we elaborate more on how to leverage this index to build a query bot that allows for querying API knowledge based on API topic issues and using our own domain-specific query language.

4 Scout-bot: API Topics Discovery and Exploration Bot

In this section we introduce *Scout-bot*, a query bot system that enables users to explore and discover APIs through API topic issues. With Scout-bot (see Figure 8), a user writes a query based on a DSL (we will present our API-Topics DSL next) and the bot returns the corresponding results containing related API resources.

In order to do so, Scout-bot first parses the user expression to identify query elements such as API names, API Topics or both, and the boolean operators that may be present in the expression. Then, results from the parser are passed on to a query generator to translate queries written in our API-Topics DSL into Elasticsearch's own DSL. Finally, the translated query is sent to Elasticsearch's search services and the results are presented back to the user. The overall query bot system is depicted in Figure 8. Next, we describe the API-topics DSL we use for writing queries in Scout-bot.

4.1 API-Topics DSL: A Domain-specific Query Language for APIs

We provide a simple yet powerful API-topics DSL that allows users to write queries based on the key entities involved in our approach: API names and API topics. We show next the queries allowed by our API-Topics DSL:

Table 4: Examples of Elasticsearch's index entries including SO questions (identified by QID) along with the *api* and *topic* mentions they relate to

SO QID	title	body	tags	api	topic
9484998	Simple FTP Client authentication?	I have written a simple FTP Client-Server code using WinSocs ...	c++, windows, winapi, authentication, ftp	winapi {winapi, win32-api, windows-api}	authentication {oauth, authorization, auth}
25885369	Windows API function CredUIPromptForWindowsCredentials also returns an error of 31	When I use the function CredUIPromptForWindowsCredentials to display a windows ...	c++, winapi	winapi {winapi, win32-api, windows-api}	security {safety, policies}, authentication {oauth, authorization, auth}
7725464	Windows security center API	I would like to access the Windows Security Center...	c++, windows, security, winapi	winapi {winapi, win32-api, windows-api}	security {safety, policies}

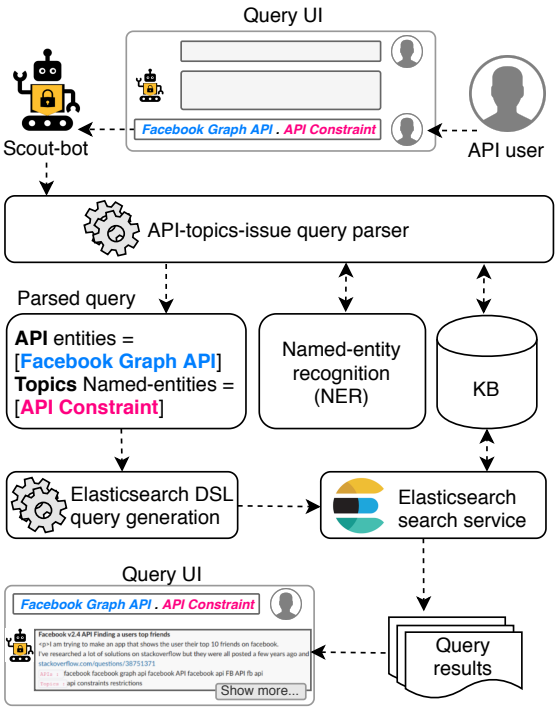

Figure 8: Overall architecture of Scout-bot

API-Name.Topic-Name This query allows users to explore a topic in the context of a given API. It suffices to provide an API and topic name separated by a "dot" (.) operator. For example, *"WinAPI"."API Security"* allows users to explore the topic of API Security in the context of Windows API.

API-Name.Topic-Name contains 'keywords' In this case, the query allows users to explore APIs and

topics based on a set of keywords. For example, a user that wants to explore authentication mechanisms in the context of API Security topic for Windows API can write a query like *"WinAPI"."API Security" contains 'Authentication'*.

API-Name.Topic-Name AND API-Name.Topic-Name This expression allows users to explore APIs using a conjunctive query. For example, if a user wants to inquire about overlapping issues of security and debugging under WinAPI, the following query can be used: *"WinAPI"."API Security" AND "WinAPI"."Debugging"* (notice that a post can be categorized under more than one topic issue).

API-Name.Topic-Name OR API-Name.Topic-Name A user can also write disjunctive queries to explore APIs and topic issues. For instance, a user exploring security issues in the context of Github API or GitLab API can write the query *"Github-API"."API Security" OR "GitLab-API"."API Security"*.

Thus, the possibility to express simple queries in terms of API names and topics (with the option to refine queries using the *contains* operator) as well as the possibility of combining them using disjunctive and conjunctive Boolean operators results in a simple query language that enables the construction of arbitrarily complex queries to support exploration and discovery of API knowledge.

4.2 Implementing and Evaluating Scout-bot

We implemented Scout-bot as a Slack[21] app. Slack is a cloud-based collaboration platform that enables users and developers to interact and communicate with each other as well as with third-party apps. We implemented both the DSL-query parser and the API Topics DSL described previously as microservices using Python Flask[22]. This serves as the webhook that connects our query bot app to our query parser and API-Topics DSL. The query bot interface app and the micro-services are deployed on an Ubuntu virtual machine hosted on Google Cloud Compute[23] service. Figure 9 shows a screenshot of Scout-bot in action where, for illustration purposes, we show only SO's main attributes (in practice, Scout-bot can show other elements from our KB model presented in Figure 5 including insights, learning resources, API documentation, among other resources).

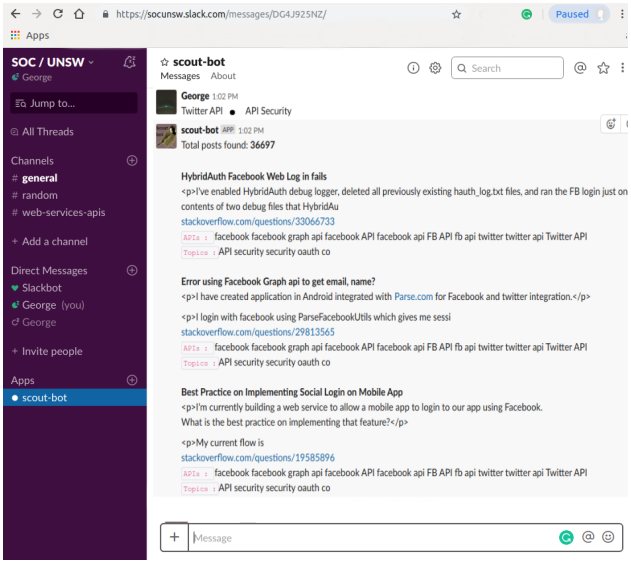

Figure 9: Implementation of Scout-bot in Slack

In order to evaluate Scout-bot, we measured its ability to retrieve relevant posts when given a query using our API-Topic DSL. To do so, we extracted, enriched and indexed a sample of approximately 62K API-related posts collected from SO. The rationale of our evaluation consists in sampling posts from these initial 62K posts, building queries that capture the essence of the API and topic being discussed in the post, and manually inspecting the returned results. The recommended minimum number of posts to be sampled was 96, considering a margin of error of 10% and a 95% confidence interval [136]. The actual number of posts sampled were 109, which allowed us to cover a number of API topic issues for at least three different

[21] https://slack.com
[22] https://flask.pocoo.org
[23] https://cloud.google.com/compute

Table 5: API-topic DSL queries based on actual questions (QID) from SO

SO QID	Formulated API-topic DSL query
13115608	facebook.usage contains 'authentication services'
2478391	youtube.usage contains 'get video title jquery'
21329250	facebook.debugging contains 'FB og image pulling'
3566018	winapi.debugging contains 'compile fatal error'
23417356	facebook.usage contains 'permanent login page access token info'
6218325	winapi.usage contains 'check if directory windows exists'
1037595	winapi.usage contains 'user interaction time'
167414	winapi.usage contains 'POSIX system file rename'
940707	winapi.usage contains 'get Win32 native APIs version'

APIs (Windows, Facebook and Youtube APIs). In order to build queries based on our API-Topic DSL we considered the posts' title and question body. Table 5 shows examples of queries built based on the sampled posts.

We evaluate the performance of Scout-bot in terms of precision [131]. We focus on how the system performs at returning relevant documents to the query in the exploration search. In this case, we follow the common information retrieval evaluation metrics Precision@K (or P@K) and R-Precision [131]. P@K helps measure the returned results at the top K number of documents. When considering the possibility of exploring more results (beyond the top K ones), it is also useful to report on the R-precision metric, which is calculated as $R\text{-}Precision = r/|rel|$. This metric measures the number of relevant documents returned by the system (r) out of the total number relevant documents ($|rel|$).

After computing the values of P@1, P@3, P@10 and R-Precision for each of the 109 formulated queries (see examples in Table 5), we proceeded with computing the mean values for these metrics across all 109 queries. The results show a performance of $\overline{P@1} = 0.99$, $\overline{P@3} = 0.96$, $\overline{P@10} = 0.52$ and $\overline{R\text{-}Precision} = 0.98$. The value 0.99 for $\overline{P@1}$ is important because users interacting with Scout-bot can have the most relevant result right in the first returned result in almost all cases. Both $\overline{P@3}$ and $\overline{R\text{-}Precision}$ also report relatively high values. In terms of $\overline{P@10}$, however, Scout-bot's performance is rather low. This shortcoming is inherent to the metric P@K when the number of relevant results is (much) lower than K [131]. For example, when the number of relevant results is 1, then the maximum value for $P@10$ is 0.1 (this is why R-Precision is also reported, i.e., to provide a more comprehensive view on performance).

5 Discussion and Future Work

This article presented a systematic mapping study that helped us identify key quality dimensions of API documentation and their coverage by existing literature. The study unveiled that completeness and understandability are two of the most researched quality dimensions within this context, where the governing concerns include incorrectness, ambiguity and incompleteness [20]. In response to these findings and challenges, this article presented an approach for indexing, exploring and discovery API community knowledge in a topic-driven manner. Our approach stems from the needs of the programming community to learn and understand APIs in an increasingly interconnected technology landscape, where APIs are considered first-class citizens. We combine our integrated KB, enriched index and simple yet powerful domain-specific query language into a useful tool, *Scout-bot*, that showcases how our solution can be seamlessly integrated into popular productivity tools such as Slack. Besides the technical advantage of integrating Scout-bot into such productivity tool, our aim was also to showcase that API community knowledge can be brought close to the toolset programmers rely on in their daily tasks, doing it in a manner that facilitates access to such knowledge without the need of switching context (e.g., opening a browser to look for online API documentations or SO Q&As). This helps bring the power of and community knowledge about APIs right into the working environment of API consumers, programmers and practitioners.

This work has its own limitations. Firstly, the API community knowledge used to build our KB is based (and mainly driven by) SO. Yet, the rationale we followed is that of relying on a reputable CQA site (SO) that converges common issues developers face when programming, in general, and, using API, in particular. Thus, by leveraging on SO, we are also leveraging both the wisdom of the crowd and the resulting curated API knowledge. Secondly, the evaluation presented in this article is focused only on the performance of

our approach in terms of (information retrieval) precision. Further and deeper studies are needed where end-users are involved to better understand the implications of our solution in learning and using APIs.

Future directions include leveraging embedding techniques for representing elements of our KB in a vector space (e.g., topics, APIs, Q&As) and the development of novel indexing and querying techniques on top of such embeddings to support semantically richer exploration and discovery. We also plan to expand our KB using Github as an additional datasource and validate our approach through extensive user studies that involve API users in realistic, production environments.

Acknowledgement

This research was done in the context of the first author's Ph.D. thesis [129] as part of the supervision of his co-authors. The work by the second author was partially supported by PRONII / CONACYT (Paraguay), Res. 148/2020.

References

[1] C. Rodríguez, M. Baez, F. Daniel, F. Casati, J. C. Trabucco, L. Canali, and G. Percannella, "REST APIs: A large-scale analysis of compliance with principles and best practices," in *ICWE'16*. Springer, 2016. doi: 10.1007/978-3-319-38791-8_2 pp. 21–39.

[2] S. Zamanirad, B. Benatallah, M. C. Barukh, F. Casati, and C. Rodriguez, "Programming bots by synthesizing natural language expressions into API invocations," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017. doi: 10.1109/ASE.2017.8115694 pp. 832–837.

[3] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, "Fogflow: Easy programming of iot services over cloud and edges for smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 696–707, 2017. doi: 10.1109/JIOT.2017.2747214

[4] Y. Zhang, L. Zhu, X. Xu, S. Chen, and A. B. Tran, "Data service API design for data analytics," in *International Conference on Services Computing*. Springer, 2018. doi: 10.1007/978-3-319-94376-3_6 pp. 87–102.

[5] F. Daniel, C. Rodríguez, S. Roy Chowdhury, H. R. Motahari Nezhad, and F. Casati, "Discovery and reuse of composition knowledge for assisted mashup development," in *Proceedings of the 21st International Conference on World Wide Web*, 2012. doi: 10.1145/2187980.2188093 pp. 493–494.

[6] C. Rodríguez, P. Silveira, F. Daniel, and F. Casati, "Analyzing compliance of service-based business processes for root-cause analysis and prediction," in *International Conference on Web Engineering*. Springer, 2010. doi: 10.1007/978-3-642-16985-4_25 pp. 277–288.

[7] P. Chatterjee, M. A. Nishi, K. Damevski, V. Augustine, L. Pollock, and N. A. Kraft, "What information about code snippets is available in different software-related documents? An exploratory study," in *SANER 2017*. IEEE, 2017. doi: 10.1109/SANER.2017.7884638 pp. 382–386.

[8] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente, "Documenting APIs with examples: Lessons learned with the APIMiner platform," in *2013 20th working conference on reverse engineering (WCRE)*. IEEE, 2013. doi: 10.1109/WCRE.2013.6671315 pp. 401–408.

[9] A. Ahmad, C. Feng, S. Ge, and A. Yousif, "A survey on mining Stack Overflow: question and answering (Q&A) community," *Data Technologies and Applications*, 2018. doi: 10.1108/DTA-07-2017-0054

[10] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced FAQs into API documentation," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014. doi: 10.1145/2591062.2591128 pp. 456–459.

[11] M. A. Saied, O. Benomar, H. Abdeen, and H. Sahraoui, "Mining multi-level API usage patterns," in *2015 IEEE 22nd international conference on software analysis, evolution, and reengineering (SANER)*. IEEE, 2015. doi: 10.1109/SANER.2015.7081812 pp. 23–32.

[12] R. B. Watson, "Development and application of a heuristic to assess trends in API documentation," in *Proceedings of the 30th ACM international conference on Design of communication*, 2012. doi: 10.1145/2379057.2379112 pp. 295–302.

[13] H. V. Nguyen, H. A. Nguyen, A. T. Nguyen, and T. N. Nguyen, "Mining interprocedural, data-oriented usage patterns in JavaScript web applications," in *Proceedings of the 36th International Conference on Software Engineering*, 2014. doi: 10.1145/2568225.2568302 pp. 791–802.

[14] M. F. Zibran, F. Z. Eishita, and C. K. Roy, "Useful, but usable? factors affecting the usability of APIs," in *2011 18th Working Conference on Reverse Engineering*. IEEE, 2011. doi: 10.1109/WCRE.2011.26 pp. 151–155.

[15] L. Thominet, "Building foundations for the crowd: minimalist author support guides for crowdsourced documentation wikis," in *Proceedings of the 33rd Annual International Conference on the Design of Communication*, 2015. doi: 10.1145/2775441.2775461 pp. 1–10.

[16] M. Kim, D. Cai, and S. Kim, "An empirical investigation into the role of API-level refactorings during software evolution," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011. doi: 10.1145/1985793.1985815 pp. 151–160.

[17] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*, 2014. doi: 10.1145/2597008.2597149 pp. 279–290.

[18] R. Watson, "Developing best practices for API reference documentation: Creating a platform to study how programmers learn new APIs," in *2012 IEEE International Professional Communication Conference*. IEEE, 2012. doi: 10.1109/IPCC.2012.6408606 pp. 1–9.

[19] A. Pawlik, J. Segal, H. Sharp, and M. Petre, "Crowdsourcing scientific software documentation: A case study of the NumPy documentation project," *Computing in Science & Engineering*, vol. 17, no. 1, pp. 28–36, 2014. doi: 10.1109/MCSE.2014.93

[20] G. Uddin and M. P. Robillard, "How API documentation fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, 2015. doi: 10.1109/MS.2014.80

[21] S. Sohan, C. Anslow, and F. Maurer, "A case study of web API evolution," in *2015 IEEE World Congress on Services*. IEEE, 2015. doi: 10.1109/SERVICES.2015.43 pp. 245–252.

[22] M. Kechagia and D. Spinellis, "Undocumented and unchecked: Exceptions that spell trouble," in *MSR'14*, 2014. doi: 10.1145/2597073.2597089 pp. 312–315.

[23] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in Stack Overflow," in *ICSM 2012*, 2012. doi: 10.1109/ICSM.2012.6405249 pp. 25–34.

[24] C. McMillan, M. Grechanik, D. Poshyvanyk, C. Fu, and Q. Xie, "Exemplar: A source code search engine for finding highly relevant applications," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1069–1087, 2011. doi: 10.1109/TSE.2011.84

[25] T. Dasgupta, M. Grechanik, E. Moritz, B. Dit, and D. Poshyvanyk, "Enhancing software traceability by automatically expanding corpora with relevant documentation," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013. doi: 10.1109/ICSM.2013.43 pp. 320–329.

[26] L. Inozemtseva, S. Subramanian, and R. Holmes, "Integrating software project resources using source code identifiers," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014. doi: 10.1145/2591062.2591108 pp. 400–403.

[27] A. L. Santos and B. A. Myers, "Design annotations to improve API discoverability," *Journal of Systems and Software*, vol. 126, pp. 17–33, 2017. doi: 10.1016/j.jss.2016.12.036

[28] G. Ajam, C. Rodriguez, and B. Benatallah, "API Topic Issues Indexing, Exploration and Discovery for API Community Knowledge," in *SLDMI-CLEI'20*, 2020. doi: 10.1109/CLEI52000.2020.00028

[29] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015. doi: 10.1016/j.infsof.2015.03.007

[30] C. E. Grant, C. P. George, V. Kanjilal, S. Nirkhiwale, J. N. Wilson, and D. Z. Wang, "A topic-based search, visualization, and exploration system," in *The Twenty-Eighth International Flairs Conference*, 2015.

[31] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013. doi: 10.5555/2999792.2999959 pp. 3111–3119.

[32] W. M. Aalst, M. Bichler, and A. Heinzl, "Robotic Process Automation," *Business & Information Systems Engineering: Vol. 60, No. 4*, 2018. doi: 10.1007/s12599-018-0542-4

[33] Q. Lu, X. Xu, Y. Liu, I. Weber, L. Zhu, and W. Zhang, "ubaas: A unified blockchain as a service platform," *Future Generation Computer Systems*, vol. 101, pp. 564–575, 2019. doi: 10.1016/j.future.2019.05.051

[34] W. Maalej and M. P. Robillard, "Patterns of knowledge in API reference documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013. doi: 10.1109/TSE.2013.12

[35] M. Ichinco, W. Y. Hnin, and C. L. Kelleher, "Suggesting API usage to novice programmers with the example guru," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017. doi: 10.1145/3025453.3025827 pp. 1105–1117.

[36] L. B. de Souza, E. C. Campos, and M. d. A. Maia, "On the extraction of cookbooks for APIs from the crowd knowledge," in *2014 Brazilian Symposium on Software Engineering*. IEEE, 2014. doi: 10.1109/SBES.2014.15 pp. 21–30.

[37] S. M. Nasehi and F. Maurer, "Unit tests as API usage examples," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010. doi: 10.1109/ICSM.2010.5609553 pp. 1–10.

[38] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, B. Russo, S. Haiduc, and M. Lanza, "CodeTube: extracting relevant fragments from software development video tutorials," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016. doi: 10.1145/2889160.2889172 pp. 645–648.

[39] G. Ajam, C. Rodriguez, and B. Benatallah, "API Topics Issues in Stack Overflow Q&As Posts: An Empirical Study," in *SLDMI-CLEI'20*, 2020. doi: 10.1109/CLEI52000.2020.00024

[40] J. Zhi, V. Garousi-Yusifoğlu, B. Sun, G. Garousi, S. Shahnewaz, and G. Ruhe, "Cost, benefits and quality of software development documentation: A systematic mapping," *Journal of Systems and Software*, vol. 99, pp. 175–198, 2015. doi: 10.1016/j.jss.2014.09.042

[41] H. Zhong and Z. Su, "Detecting API documentation errors," in *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, 2013. doi: 10.1145/2544173.2509523 pp. 803–816.

[42] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: understanding the decisions of open source contributors," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010. doi: 10.1145/1882291.1882312 pp. 127–136.

[43] D. Ko, K. Ma, S. Park, S. Kim, D. Kim, and Y. Le Traon, "API document quality for resolving deprecated APIs," in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 2. IEEE, 2014. doi: 10.1109/APSEC.2014.87 pp. 27–30.

[44] R. Watson, M. Stamnes, J. Jeannot-Schroeder, and J. H. Spyridakis, "API documentation and software community values: a survey of open-source API documentation," in *Proceedings of the 31st ACM international conference on Design of communication*, 2013. doi: 10.1145/2507065.2507076 pp. 165–174.

[45] L. Shi, H. Zhong, T. Xie, and M. Li, "An empirical study on evolution of API documentation," in *International Conference on Fundamental Approaches To Software Engineering*. Springer, 2011. doi: 10.1007/978-3-642-19811-3_29 pp. 416–431.

[46] S. Sohan, F. Maurer, C. Anslow, and M. P. Robillard, "A study of the effectiveness of usage examples in REST API documentation," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2017. doi: 10.1109/VLHCC.2017.8103450 pp. 53–61.

[47] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *2011 international symposium on empirical software engineering and measurement*. IEEE, 2011. doi: 10.1109/ESEM.2011.36 pp. 275–284.

[48] S. Keele *et al.*, "Guidelines for performing systematic literature reviews in software engineering," Technical report, Ver. 2.3 EBSE Technical Report. EBSE, Tech. Rep., 2007.

[49] M. Gusenbauer, "Google scholar to overshadow them all? comparing the sizes of 12 academic search engines and bibliographic databases," *Scientometrics*, vol. 118, no. 1, pp. 177–214, 2019. doi: 10.1007/s11192-018-2958-5

[50] A. Martín-Martín, E. Orduna-Malea, M. Thelwall, and E. D. López-Cózar, "Google scholar, web of science, and scopus: A systematic comparison of citations in 252 subject categories," *Journal of informetrics*, vol. 12, no. 4, pp. 1160–1177, 2018. doi: 10.1016/j.joi.2018.09.002

[51] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014. doi: 10.1145/2601248.2601268 pp. 1–10.

[52] J. W. Creswell, *Educational research: Planning, conducting, and evaluating quantitative.* Prentice Hall Upper Saddle River, NJ, 2002.

[53] J. Kim, S. Lee, S.-w. Hwang, and S. Kim, "Adding examples into Java documents," in *2009 IEEE/ACM International Conference on Automated Software Engineering.* IEEE, 2009. doi: 10.1109/ASE.2009.39 pp. 540–544.

[54] M. Flatt, E. Barzilay, and R. B. Findler, "Scribble: Closing the book on ad hoc documentation tools," *ACM Sigplan Notices*, vol. 44, no. 9, pp. 109–120, 2009. doi: 10.1145/1596550.1596569

[55] J. Kim, S. Lee, S.-W. Hwang, and S. Kim, "Enriching documents with examples: A corpus mining approach," *ACM Transactions on Information Systems (TOIS)*, vol. 31, no. 1, pp. 1–27, 2013. doi: 10.1145/2414782.2414783

[56] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live API documentation," in *Proceedings of the 36th International Conference on Software Engineering*, 2014. doi: 10.1145/2568225.2568313 pp. 643–652.

[57] L. W. Mar, Y.-C. Wu, and H. C. Jiau, "Recommending proper API code examples for documentation purpose," in *2011 18th Asia-Pacific Software Engineering Conference.* IEEE, 2011. doi: 10.1109/APSEC.2011.18 pp. 331–338.

[58] G. Petrosyan, M. P. Robillard, and R. De Mori, "Discovering information explaining API types using text classification," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015. doi: 10.1109/ICSE.2015.97 pp. 869–879.

[59] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from Stack Overflow," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE).* IEEE, 2016. doi: 10.1145/2884781.2884800 pp. 392–403.

[60] U. Dekel and J. D. Herbsleb, "Improving API documentation usability with knowledge pushing," in *2009 IEEE 31st International Conference on Software Engineering.* IEEE, 2009. doi: 10.1109/ICSE.2009.5070532 pp. 320–330.

[61] C. Gao, J. Wei, H. Zhong, and T. Huang, "Inferring data contract for web-based API," in *2014 IEEE International Conference on Web Services.* IEEE, 2014. doi: 10.1109/ICWS.2014.22 pp. 65–72.

[62] M. A. Saied, H. Sahraoui, and B. Dufour, "An observational study on API usage constraints and their documentation," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER).* IEEE, 2015. doi: 10.1109/SANER.2015.7081813 pp. 33–42.

[63] M. Monperrus, M. Eichberg, E. Tekes, and M. Mezini, "What should developers be aware of? An empirical study on the directives of API documentation," *Empirical Software Engineering*, vol. 17, no. 6, pp. 703–737, 2012. doi: 10.1007/s10664-011-9186-4

[64] M. P. Robillard, "What makes APIs hard to learn? Answers from developers," *IEEE software*, vol. 26, no. 6, pp. 27–34, 2009. doi: 10.1109/MS.2009.193

[65] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web APIs on the world wide web," in *2010 eighth ieee european conference on web services.* IEEE, 2010. doi: 10.1109/ECOWS.2010.9 pp. 107–114.

[66] D. Hou and L. Li, "Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions," in *2011 IEEE 19th International Conference on Program Comprehension*. IEEE, 2011. doi: 10.1109/ICPC.2011.21 pp. 91–100.

[67] M. P. Robillard and R. Deline, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011. doi: 10.1007/s10664-010-9150-8

[68] B. Dagenais and M. P. Robillard, "Recovering traceability links between an API and its learning resources," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012. doi: 10.1109/ICSE.2012.6227207 pp. 47–57.

[69] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow," *Georgia Institute of Technology, Tech. Rep*, vol. 11, 2012.

[70] R. Silva, C. Roy, M. Rahman, K. Schneider, K. Paixao, and M. Maia, "Recommending comprehensive solutions for programming tasks by mining crowd knowledge," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019. doi: 10.1109/ICPC.2019.00054 pp. 358–368.

[71] Y. Zhou, C. Wang, X. Yan, T. Chen, S. Panichella, and H. C. Gall, "Automatic detection and repair recommendation of directive defects in Java API documentation," *IEEE Transactions on Software Engineering*, 2018. doi: 10.1109/TSE.2018.2872971

[72] P. Sun, C. Brown, I. Beschastnikh, and K. T. Stolee, "Mining specifications from documentation using a crowd," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019. doi: 10.1109/SANER.2019.8668025 pp. 275–286.

[73] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, "ARENA: an approach for the automated generation of release notes," *IEEE Transactions on Software Engineering*, vol. 43, no. 2, pp. 106–127, 2016. doi: 10.1109/TSE.2016.2591536

[74] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, "Software documentation issues unveiled," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019. doi: 10.1109/ICSE.2019.00122 pp. 1199–1210.

[75] A. Cummaudo, R. Vasa, and J. Grundy, "What should I document? A preliminary systematic mapping study into API documentation knowledge," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019. doi: 10.1109/ESEM.2019.8870148 pp. 1–6.

[76] J. Zhang, H. Jiang, Z. Ren, T. Zhang, and Z. Huang, "Enriching API Documentation with Code Samples and Usage Scenarios from Crowd Knowledge," *IEEE Transactions on Software Engineering*, 2019. doi: 10.1109/TSE.2019.2919304

[77] L. Cai, H. Wang, Q. Huang, X. Xia, Z. Xing, and D. Lo, "BIKER: a tool for Bi-information source based API method recommendation," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019. doi: 10.1145/3338906.3341174 pp. 1075–1079.

[78] K. Thayer, "Using Program Analysis to Improve API Learnability," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018. doi: 10.1145/3230977.3231009 pp. 292–293.

[79] M. Meng, S. M. Steinhardt, and A. Schubert, "Optimizing API Documentation: Some Guidelines and Effects," in *Proceedings of the 38th ACM International Conference on Design of Communication*, 2020. doi: 10.1145/3380851.3416759 pp. 1–11.

[80] X. Ren, J. Sun, Z. Xing, X. Xia, and J. Sun, "Demystify official API usage directives with crowdsourced API misuse scenarios, erroneous code examples and patches," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020. doi: 10.1145/3377811.3380430 pp. 925–936.

[81] J. Yang, E. Wittern, A. T. Ying, J. Dolby, and L. Tan, "Towards extracting web API specifications from documentation," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 2018. doi: 10.1145/3196398.3196411 pp. 454–464.

[82] H. Phan, H. A. Nguyen, T. N. Nguyen, and H. Rajan, "Statistical learning for inference between implementations and documentation," in *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*. IEEE, 2017. doi: 10.1109/ICSE-NIER.2017.9 pp. 27–30.

[83] A. Head, C. Sadowski, E. Murphy-Hill, and A. Knight, "When not to comment: questions and tradeoffs with API documentation for C++ projects," in *Proceedings of the 40th International Conference on Software Engineering*, 2018. doi: 10.1145/3180155.3180176 pp. 643–653.

[84] J. Li, A. Sun, Z. Xing, and L. Han, "API Caveat Explorer–Surfacing Negative Usages from Practice: An API-oriented Interactive Exploratory Search System for Programmers," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018. doi: 10.1145/3209978.3210170 pp. 1293–1296.

[85] S. Inzunza, R. Juárez-Ramírez, and S. Jiménez, "API documentation: A Conceptual Evaluation Model," in *World Conference on Information Systems and Technologies*. Springer, 2018. doi: 10.1007/978-3-319-77712-2_22 pp. 229–239.

[86] Y. Tao, J. Jiang, Y. Liu, Z. Xu, and S. Qin, "Understanding Performance Concerns in the API Documentation of Data Science Libraries," in *The 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020. doi: 10.1145/3324884.3416543

[87] Y. Tian, F. Thung, A. Sharma, and D. Lo, "APIBot: question answering bot for API documentation," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017. doi: 10.1109/ASE.2017.8115628 pp. 153–158.

[88] K. W. Nafi, B. Roy, C. K. Roy, and K. A. Schneider, "CroLSim: Cross Language Software Similarity Detector Using API Documentation," in *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2018. doi: 10.1109/SCAM.2018.00023 pp. 139–148.

[89] J. Li, A. Sun, and Z. Xing, "To Do or Not To Do: Distill crowdsourced negative caveats to augment API documentation," *Journal of the Association for Information Science and Technology*, vol. 69, no. 12, pp. 1460–1475, 2018. doi: 10.1002/asi.24067

[90] G. Uddin, F. Khomh, and C. K. Roy, "Towards crowd-sourced API documentation," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019. doi: 10.1109/ICSE-Companion.2019.00129 pp. 310–311.

[91] H. E. Salman, "API library-based identification and documentation of usage patterns," *International Journal of Computer Applications in Technology*, vol. 58, no. 1, pp. 63–79, 2018. doi: 10.1504/IJCAT.2018.094065

[92] S. Thummalapenta and T. Xie, "Alattin: Mining alternative patterns for detecting neglected conditions," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2009. doi: 10.1109/ASE.2009.72 pp. 283–294.

[93] R. P. Buse and W. Weimer, "Synthesizing API usage examples," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012. doi: 10.1109/ICSE.2012.6227140 pp. 782–792.

[94] C. Parnin and C. Treude, "Measuring API documentation on the web," in *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, 2011. doi: 10.1145/1984701.1984706 pp. 25–30.

[95] L. Wang, Y. Zou, L. Fang, B. Xie, and F. Yang, "An exploratory study of API usage examples on the web," in *2012 19th Asia-Pacific Software Engineering Conference*, vol. 1. IEEE, 2012. doi: 10.1109/APSEC.2012.122 pp. 396–405.

[96] L. Guerrouj, D. Bourque, and P. C. Rigby, "Leveraging informal documentation to summarize classes and methods in context," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015. doi: 10.1109/ICSE.2015.212 pp. 639–642.

[97] X. Ren, Z. Xing, X. Xia, G. Li, and J. Sun, "Discovering, explaining and summarizing controversial discussions in community Q&A sites," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019. doi: 10.1109/ASE.2019.00024 pp. 151–162.

[98] R. B. Watson, "Improving software API usability through text analysis: A case study," in *2009 IEEE International Professional Communication Conference.* IEEE, 2009. doi: 10.1109/IPCC.2009.5208679 pp. 1–7.

[99] M. Bruch, M. Mezini, and M. Monperrus, "Mining subclassing directives to improve framework reuse," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010).* IEEE, 2010. doi: 10.1109/MSR.2010.5463347 pp. 141–150.

[100] U. Dekel and J. D. Herbsleb, "Reading the documentation of invoked API functions in program comprehension," in *2009 IEEE 17th International Conference on Program Comprehension.* IEEE, 2009. doi: 10.1109/ICPC.2009.5090040 pp. 168–177.

[101] Y. Zhou, R. Gu, T. Chen, Z. Huang, S. Panichella, and H. Gall, "Analyzing APIs documentation and code to detect directive defects," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE).* IEEE, 2017. doi: 10.1109/ICSE.2017.11 pp. 27–37.

[102] 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).* IEEE, 2018. doi: 10.1109/ICSME.2018.00028 pp. 183–193.

[103] Y.-C. Wu, L. W. Mar, and H. C. Jiau, "CoDocent: Support API usage with code example and API documentation," in *2010 Fifth International Conference on Software Engineering Advances.* IEEE, 2010. doi: 10.1109/ICSEA.2010.28 pp. 135–140.

[104] S. Sohan, C. Anslow, and F. Maurer, "Automated example oriented REST API documentation at Cisco," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP).* IEEE, 2017. doi: 10.1109/ICSE-SEIP.2017.11 pp. 213–222.

[105] M. Michalski, P. Kosko, D. Juszczak, and H. Kwon, "EWIDL: Single-Source Web API Documentation Management System," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 2020. doi: 10.1109/ICSME46990.2020.00081 pp. 723–726.

[106] S. Lee, R. Wu, S.-C. Cheung, and S. Kang, "Automatic detection and update suggestion for outdated API names in documentation," *IEEE Transactions on Software Engineering*, 2019. doi: 10.1109/TSE.2019.2901459

[107] S. M. Sohan, C. Anslow, and F. Maurer, "Spyrest: Automated restful API documentation using an HTTP proxy server (N)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 2015. doi: 10.1109/ASE.2015.52 pp. 271–276.

[108] M. Hosono, H. Washizaki, Y. Fukazawa, and K. Honda, "An Empirical Study on the Reliability of the Web API Document," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC).* IEEE, 2018. doi: 10.1109/APSEC.2018.00103 pp. 715–716.

[109] F. F. Correia, A. Aguiar, H. S. Ferreira, and N. Flores, "Patterns for consistent software documentation," in *Proceedings of the 16th Conference on Pattern Languages of Programs*, 2009. doi: 10.1145/1943226.1943241 pp. 1–7.

[110] P. J. Danielsen and A. Jeffrey, "Validation and interactivity of web API documentation," in *2013 IEEE 20th International Conference on Web Services.* IEEE, 2013. doi: 10.1109/ICWS.2013.76 pp. 523–530.

[111] H. Zhong, L. Zhang, T. Xie, and H. Mei, "Inferring resource specifications from natural language API documentation," in *2009 IEEE/ACM International Conference on Automated Software Engineering.* IEEE, 2009. doi: 10.1109/ASE.2009.94 pp. 307–318.

[112] Y. Zhou, X. Yan, T. Chen, S. Panichella, and H. Gall, "DRONE: a tool to detect and repair directive defects in Java APIs documentation," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion).* IEEE, 2019. doi: 10.1109/ICSE-Companion.2019.00052 pp. 115–118.

[113] S. Lee, S. Lee, S. Lim, J. Jung, S. Choi, N. Kim, and J.-B. Lee, "An API Design Process in Terms of Usability: A Case Study on Building More Usable APIs for Smart TV Platform," in *2014 IEEE 38th International Computer Software and Applications Conference Workshops.* IEEE, 2014. doi: 10.1109/COMPSACW.2014.95 pp. 567–571.

[114] F. Logozzo, "Practical specification and verification with code contracts," *ACM SIGAda Ada Letters*, vol. 33, no. 3, pp. 7–8, 2013. doi: 10.1145/2658982.2534188

[115] A. Head, C. Appachu, M. A. Hearst, and B. Hartmann, "Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2015. doi: 10.1109/VLHCC.2015.7356972 pp. 3–12.

[116] M. A. Saied, H. Abdeen, O. Benomar, and H. Sahraoui, "Could we infer unordered API usage patterns only using the library source code?" in *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 2015. doi: 10.1109/ICPC.2015.16 pp. 71–81.

[117] D. S. Eisenberg, J. Stylos, A. Faulring, and B. A. Myers, "Using association metrics to help users navigate API documentation," in *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2010. doi: 10.1109/VLHCC.2010.13 pp. 23–30.

[118] D. S. Eisenberg, J. Stylos, and B. A. Myers, "Apatite: A new interface for exploring APIs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010. doi: 10.1145/1753326.1753525 pp. 1331–1334.

[119] M. Pradel and T. R. Gross, "Automatic generation of object usage specifications from large method traces," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2009. doi: 10.1109/ASE.2009.60 pp. 371–382.

[120] M. Horie and S. Chiba, "Tool support for crosscutting concerns of API documentation," in *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, 2010. doi: 10.1145/1739230.1739242 pp. 97–108.

[121] S. Spiza and S. Hanenberg, "Type names without static type checking already improve the usability of APIs (as long as the type names are correct) an empirical study," in *Proceedings of the 13th international conference on Modularity*, 2014. doi: 10.1145/2577080.2577098 pp. 99–108.

[122] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar, "Inferring method specifications from natural language API descriptions," in *2012 34th international conference on software engineering (ICSE)*. IEEE, 2012. doi: 10.1109/ICSE.2012.6227137 pp. 815–825.

[123] A. P. Bhardwaj, D. Luciano, and S. R. Klemmer, "Redprint: integrating API specific "instant example" and "instant documentation" display interface in IDEs," in *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*, 2011. doi: 10.1145/2046396.2046408 pp. 21–22.

[124] A. J. Ko and Y. Riche, "The role of conceptual knowledge in API usability," in *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2011. doi: 10.1109/VLHCC.2011.6070395 pp. 173–176.

[125] C. González-Mora, C. Barros, I. Garrigós, J. Zubcoff, E. Lloret, and J.-N. Mazón, "Applying Natural Language Processing Techniques to Generate Open Data Web APIs Documentation," in *International Conference on Web Engineering*. Springer, 2020. doi: 10.1007/978-3-030-50578-3_28 pp. 416–432.

[126] T. D. Nguyen, A. T. Nguyen, H. D. Phan, and T. N. Nguyen, "Exploring API embedding for API usages and applications," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017. doi: 10.1109/ICSE.2017.47 pp. 438–449.

[127] T. Nguyen, N. Tran, H. Phan, T. Nguyen, L. Truong, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Complementing global and local contexts in representing API descriptions to improve API retrieval tasks," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018. doi: 10.1145/3236024.3236036 pp. 551–562.

[128] F. M. Delfim, K. V. Paixão, D. Cassou, and M. de Almeida Maia, "Redocumenting APIs with crowd knowledge: A coverage analysis based on question types," *Journal of the Brazilian Computer Society*, vol. 22, no. 1, p. 9, 2016. doi: 10.1186/s13173-016-0049-0

[129] G. Ajam, "Quality of Application Programming Interfaces Documentation and Topics Issues Exploration." Ph.D. Thesis, School of Computer Science and Engineering, Faculty of Engineering, UNSW Sydney, 2019.

[130] V. Efstathiou, C. Chatzilenas, and D. Spinellis, "Word embeddings for the software engineering domain," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018. doi: 10.1145/3196398.3196448 pp. 38–41.

[131] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press, 2008.

[132] S. Mumtaz, C. Rodriguez, B. Benatallah, M. Al-Banna, and S. Zamanirad, "Learning Word Representation for the Cyber Security Vulnerability Domain," in *The 2020 International Joint Conference on Neural Networks (IJCNN 2020)*, 2020. doi: 10.1109/IJCNN48605.2020.9207140

[133] S. Mumtaz, C. Rodriguez, and B. Benatallah, "Expert2vec: Experts representation in community question answering for question routing," in *International Conference on Advanced Information Systems Engineering*. Springer, 2019. doi: 10.1007/978-3-030-21290-2_14 pp. 213–229.

[134] C. Chen, Z. Xing, and X. Wang, "Unsupervised software-specific morphological forms inference from informal discussions," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017. doi: 10.1109/ICSE.2017.48 pp. 450–461.

[135] M. Seshadri, S. Jayakymar, A. Bansal, and M. G. Gurno, "API Recognition in Stack Overflow Posts," http://madhavanseshadri.com/static/Stackoverflow.pdf.

[136] G. D. Israel, "Determining sample size," University of Florida Cooperative Extension Service, Institute of Food and Agricultural Sciences Extension. https://www.tarleton.edu/academicassessment/documents/Samplesize.pdf, 1992.