What should I document? A preliminary systematic mapping study into API documentation knowledge

Alex Cummaudo
Applied Artificial Intelligence Institute
Deakin University
Geelong, Australia
ca@deakin.edu.au

Rajesh Vasa
Applied Artificial Intelligence Institute
Deakin University
Geelong, Australia
rajesh.vasa@deakin.edu.au

John Grundy
Faculty of Information Technology
Monash University
Clayton, Australia
john.grundy@monash.edu

***Abstract*—*Background:* Good API documentation facilitates the development process, improving productivity and quality. While the topic of API documentation quality has been of interest for the last two decades, there have been few studies to map the specific constructs needed to create a good document. In effect, we still need a structured taxonomy that captures such knowledge systematically. *Aims:* This study reports emerging results of a systematic mapping study. We capture key conclusions from previous studies that assess API documentation quality, and synthesise the results into a single framework. *Method:* By conducting a systematic review of 21 key works, we have developed a five dimensional taxonomy based on 34 categorised weighted recommendations. *Results:* All studies utilise field study techniques to arrive at their recommendations, with seven studies employing some form of interview and questionnaire, and four conducting documentation analysis. The taxonomy we synthesise reinforces that usage description details (code snippets, tutorials, and reference documents) are generally highly weighted as helpful in API documentation, in addition to design rationale and presentation. *Conclusions:* We propose extensions to this study aligned to developer utility for each of the taxonomy's categories.**

***Index Terms*—API, documentation, DevX, systematic mapping study, taxonomy**

I. Introduction

Improving the quality of API documentation is highly valuable to the software development process; good documentation facilitates productivity and thus quality is better engineered into the system [1]. Where an application developer integrates new pieces of functionality (via APIs) into a system, their productivity is affected either by inadequate skills (*"I've never used an API like this, so must learn from scratch"*) or, where their skills are adequate, an imbalanced cognitive load that causes excessive context switching (*"I have the skills for this, but am confused or misunderstand"*). In the latter case, what causes this confusion and how to mitigate it via improved API documentation is an area that has been explored; prior studies have provided recommendations based on both qualitative and quantitative analysis of developer's opinions. These recommendations and guidelines propose ways by which developers, managers and solution architects can construct systems better.

However, to date there has been little attempt to systematically capture this knowledge about API documentation from various studies into a readily accessible, consolidated format, that assists API designers to prepare better documentation. While previous works have covered certain aspects of API usage, many have lacked a systematic review of literature and do not offer a taxonomy to consolidate these guidelines together. For example, some studies have considered the technical implementation improving API usability or tools to generate (or validate) API documentation from its source code (e.g., [2–4]). However, there still lacks a consolidated effort to capture the knowledge and artefacts best suited to *manually write* API documentation. This paper presents outcomes from a preliminary work to address this gap and offers two key contributions:

- a systematic mapping study (SMS) consisting of 21 studies that capture what knowledge or artefacts should be contained within API documentation; and,
- a structured taxonomy based on the consolidated recommendations of these 21 studies.

After performing our SMS on what API knowledge should be captured in documentation—to assist API designers—we propose a five dimensional taxonomy consisting of: (1) Usage Description; (2) Design Rationale; (3) Domain Concepts; (4) Support Artefacts; and (5) Documentation Presentation.

This paper is structured as thus: section II presents related work in the area; section III is divided into two subsections, the first describing how primary sources were selected in a SMS, with the second describing the development of our taxonomy from these sources; section IV presents our primary studies and our proposed taxonomy; section V describes the threats to validity of this work and section VI provides concluding remarks and the future directions of this study.

II. Related Work

SMSs have previously been explored in the area of API usability and developer experience. Nybom et al. [2] recently performed an SMS on 36 API documentation generation tools and approaches. An analysis is presented of the API documentation-related tools developed, what kinds of documentation is generated by them, and the dependencies they require to generate this documentation. Their findings highlight a recent effort on the development of API documentation by producing example code snippets and/or templates on how

978-1-7281-2968-6/19/$31.00 ©2019 Crown

to use the API or bootstrap developers to begin using the APIs. A secondary focus is closely followed by tools that produce natural language descriptions that can be produced within developer documentation. However, Nybom et al. produce an SMS on the types of *tooling* that exists to assist in producing and validating API documentation. While this is a systematic study with key insights into the types of tooling produced, there is still a gap for an SMS in what *guidelines* have been produced by the literature in developing natural-language documentation itself, which our work has addressed.

Watson [3] performed a heuristic assessment of 11 high-level universal design elements of API documentation against 35 popular APIs. He demonstrated that many of these popular APIs fail to grasp even the basic of these elements; for example, 25% of the documentation sets did not provide any basic overview documentation. However, the heuristic used within this study consists of just 11 elements and is based on only three seminal works. Our work extends these heuristics and structures them into a consolidated, hierarchical taxonomy using a systematic taxonomy development method for SE.

A taxonomy of knowledge patterns within API reference documentation by Maalej and Robillard [4] classified 12 distinct knowledge types. Evaluation of the taxonomy against JDK 6 and .NET 4.0 showed that, while functionality and structure of the API is well-communicated, core concepts and rationale about the API are quite rare to find. Moreover, they demonstrated that low-value 'non-information' (documentation that provides uninformative boilerplate text with no insight into the API at all) is substantially present in the documentation of methods and fields in these APIs. Their findings recommend that developers factor their 12 distinct knowledge types into the process of code documentation and prevent producing low-value documentation. The development of their taxonomy consisted of questions to model knowledge and information, thereby capturing the reason about disparate information units independent to context; a key difference to this paper is the systematic taxonomy approach we have utilised.

III. Method

Our taxonomy development consisted of two phases. Firstly, we conducted an SMS to identify and analyse API documentation studies, following the guidelines of Kitchenham and Charters [5] and Petersen et al. [6]. Following this, we followed the software engineering (SE) taxonomy development method devised by Usman et al. [7] on our findings from the SMS.

A. Systematic Mapping Study

1) Research Questions (RQs): To guide our SMS, we developed the following RQs:

- **RQ1** What knowledge do API documentation studies contribute?
- **RQ2** How is API documentation studied?

The intent behind RQ1 is to collate as much of the insight provided by the literature on how API providers should best document their work. This helped us shape and form the taxonomy provided in section IV. RQ2 addresses methodologies by which these studies come to these conclusions to identify gaps in literature where future studies can potentially focus.

2) Automatic Filtering: Informed by similar previous studies in SE [8, 7, 9], we begin by defining the SWEBOK [10] knowledge areas (KAs) to assist in the search and mapping process of an SMS. Our search query was built using related KAs, relevant synonyms, and the term 'software engineering' (for comprehensiveness) all joined with the OR operator. Due to the lack of a standard definition of an API, we include the terms: 'API' and its expanded term; software library, component and framework; and lastly SDK and its expanded term. These too were joined with the OR operator, appended with an AND. Lastly, the term 'documentation' was appended with an AND. Our final search string was:

> ("software design" **OR** "software architecture" **OR** "software construction" **OR** "software development" **OR** "software maintenance" **OR** "software engineering process" **OR** "software process" **OR** "software lifecycle" **OR** "software methods" **OR** "software quality" **OR** "software engineering professional practice" **OR** "software engineering") **AND** (api **OR** "application programming interface" **OR** "software library" **OR** "software component" **OR** "software framework" **OR** sdk **OR** "software development kit") **AND** (documentation)

TABLE I
Summary of our search results and publication types

Publication type	WoS	C/I	Scopus	Total
Conference Paper	27	442	2353	2822
Journal Article	41	127	1236	1404
Book	23	17	224	264
Other	0	5	6	11
Total	91	591	3819	4501

The query was then executed on all available metadata (title, abstract and keywords) on three primary sources to search for relevant studies in May 2019. Web of Science[1] (WoS), Compendex/Inspec[2] (C/I) and Scopus[3] were chosen due to their relevance in SE literature (containing the IEEE, ACM, Springer and Elsevier databases) and their ability to support advanced queries [11, 5]. A total 4,501 results[4] were found, with 514 being duplicates. Table I displays our results in further detail (duplicates not omitted).

3) Manual Filtering: A follow-up manual filtering to select primary studies was performed on the 4,501 results using the following inclusion criteria (IC) and exclusion criteria (EC):

- **IC1** Studies must be relevant to API documentation: specifically, we exclude studies that deal with improving the technical API usability (e.g., improved usage patterns);
- **IC2** Studies must propose new knowledge or recommendations to document APIs;

[1]http://apps.webofknowlegde.com last accessed 23 May 2019.
[2]http://www.engineeringvillage.com last accessed 23 May 2019.
[3]http://www.scopus.com last accessed 23 May 2019.
[4]Raw results can be located at http://bit.ly/2KxBLs4

IC3 Studies must be relevant to SE as defined in SWEBOK;

EC1 Studies where full-text is not accessible through standard institutional databases;

EC2 Studies that do not propose or extend how to improve the official, natural language documentation of an API;

EC3 Studies proposing a third-party tool to enhance existing documentation or generate new documentation using data mining (i.e., not proposing strategies to improve official documentation);

EC4 Studies not written in English;

EC5 Studies not peer-reviewed.

After exporting metadata of search results to a spreadsheet, a three-phase curation process was conducted. The first author read the publication source (to omit non-SE papers quickly), author keywords and title of all 4,501 studies (514 that were duplicates), and abstract. As we considered multiple databases, some studies were repeated. However, the DOIs and titles were sorted and reviewed, retaining only one copy of the paper from a single database. Moreover, as there was no limit to the year range in our query, some studies were republished in various venues. These, too, were handled with title similarity matching, wherein only the first paper was considered. Where the inclusion or exclusion criteria could not be determined from the abstract alone, the paper was automatically shortlisted. In total, 133 studies were shortlisted to the second phase. We rejected 427 studies that were unrelated to SE, 3,235 were not directly related to documenting APIs (e.g., to enhance coding techniques that improve the overall developer usability of the API), 182 proposed new tools to enhance API documentation or used machine learning to mine developer's discussion of APIs, and 10 were not in English.

The shortlisted studies were then re-evaluated by re-reading the abstract, the introduction and conclusion. Performing this second phase removed a further 64 studies that were on API usability or non API-related documentation (e.g., code commenting); we further refined our exclusion criteria to better match the research outcomes of this goal (chiefly including the word 'natural language' documentation in EC2) which removed studies focused to improve technical documentation of APIs such as data types and communication schemas. Additionally, 26 studies were removed as they were related to introducing new tools (EC3), 3 were focused on tools to mine API documentation, 7 studies where no recommendations were provided, 2 further duplicate studies, and a further 10 studies where the full text was not available, not peer reviewed or in English. Books are commonly not peer-reviewed (EC5), however no books were shortlisted within these results. **This resulted in 21 primary studies for further analysis. The mapping of primary study identifiers to references S1–21 can be found at http://bit.ly/2MtsIuE**.

Intra-rater reliability of our 133 shortlisted papers was tested using the test-retest approach [5] by re-evaluating a random sample of 10% (13 total) of the studies shortlisted a week after initial studies were shortlisted. Using the Cohen's kappa coefficient as a metric for reliability, $\kappa = 0.7547$, indicating substantial agreement [12].

TABLE II
DATA EXTRACTION FORM

Data item(s)	Description
Citation metadata	Title, author(s), years, publication venue, publication type
Key recommendation(s)	As per IC2, the study must propose at least one recommendation on what should be captured in API documentation
Evaluation method	Did the authors evaluate their recommendations? If so, how?
Primary technique	The primary technique used to devise the recommendation(s)
Secondary technique	As above, if a second study was conducted
Tertiary technique	As above, if a third study was conducted
Research type	The research type employed in the study as defined by Wieringa and Heerkens's taxonomy

4) Data Extraction: Of the 21 primary studies, we conducted abstract key-wording adhering to Petersen et al.'s guidelines [6] to develop a classification scheme. An initial set of keywords were applied for each paper in terms of their methodologies and research approaches (RQ2), based on an existing classification schema by Wieringa and Heerkens [13]: evaluation, validation, personal experience and philosophical papers.

After all primary studies had been assigned keywords, we noticed that **all papers used field study techniques**, and thus we consolidated these keywords using Singer et al.'s framework of SE field study techniques [14]. Singer et al. captures both study techniques *and* methods to collect data within the one framework, namely: *direct techniques*, including brainstorming and focus groups, interviews and questionnaires, conceptual modelling, work diaries, think-aloud sessions, shadowing and observation, participant observation; *indirect techniques*, including instrumenting systems, fly-on-the-wall; and *independent techniques*, including analysis of work databases, tool use logs, documentation analysis, and static and dynamic analysis.

Table II describes our data extraction form, which was used to collect relevant data from each paper. Figure 1 maps each study to one (or more, if applicable) of methodologies plotted against Wieringa and Heerkens's research approaches.

B. Development of the Taxonomy

Usman et al. concludes that a majority of SE taxonomies are developed in an ad-hoc way [7], and proposes a systematic approach to develop taxonomies in SE that extends previous efforts by including lessons learned from more mature fields. In this subsection, we outline the 4 phases and 13 steps taken to develop our taxonomy based on Usman et al.'s technique.

1) Planning phase: The planning phase of the technique involves the following six steps:

(1) *defining the SE KA*: The software engineering KA, as defined by the SWEBOK, is software construction;

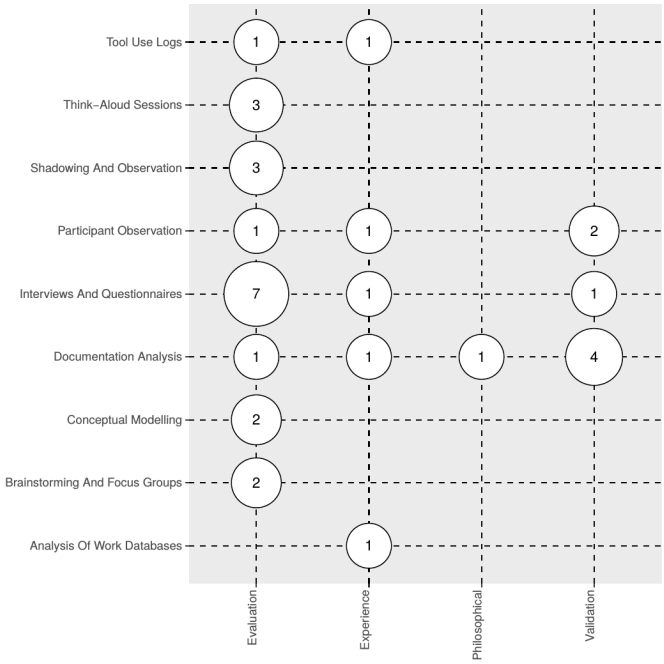

Fig. 1. Systematic map - field study technique vs research type

(2) *defining the objective*: The main objective of the proposed taxonomy is to define a set of categories that classifies different facets of natural-language API *documentation* knowledge (not API *usability* knowledge) as reported in existing literature;

(3) *defining the subject matter*: The subject matter of our proposed taxonomy is documentation artefacts of APIs;

(4) *defining the classification structure*: The classification structure of our proposed taxonomy is *hierarchical*;

(5) *defining the classification procedure*: The procedure used to classify the documentation artefacts is qualitative;

(6) *defining the data sources*: The basis of the taxonomy is derived from field study techniques (see section III-A4).

2) Identification and extraction phase: The second phase of the taxonomy development involves **(7)** *extracting all terms and concepts* from relevant literature, as selected from our 21 primary studies. These terms are then consolidated by **(8)** *performing terminology control*, as some terms may refer to different concepts and vice-versa.

3) Design phase: The design phase identified the core dimensions and categories within the extracted data items. The first step is to **(9)** *identify and define taxonomy dimensions*; for this study we utilised a bottom-up approach to identify the dimensions, i.e. extracting the categories first and then nominating which dimensions these categories fit into using an iterative approach. As a bottom-up approach was utilised, step (9) also encompassed the second stage of the design phase, which is to **(10)** *identify and describe the categories* of each dimension. Thirdly, we **(11)** *identify and describe relationships* between dimensions and categories, which can be skipped if the relationships are too close together, as is the case of our grouping technique which allows for new dimensions and categories to be added. The last step in this phase is to **(12)** *define guidelines for using and updating the taxonomy*, however as this taxonomy still an emerging result, guidelines to update and use the taxonomy are anticipated future work.

4) Validation phase: In the final phase of taxonomy development, taxonomy designers must **(13)** *validate the taxonomy* to assess its usefulness. Ideally, this is done by applying the taxonomy heuristically against developers or real-world case-studies. This remains a plan for future work (see section V).

IV. Taxonomy

Our taxonomy consists of five dimensions (labelled A–E) that respectively cover: **[A] Usage Description** on *how* to use the API for the developer's intended use case; **[B] Design Rationale** on *when* the developer should choose this API for a particular use case; **[C] Domain Concepts** of the domain behind the API to understand *why* this API should be chosen for this domain; **[D] Support Artefacts** that describe *what* additional documentation the API provides; and **[E] Documentation Presentation** to help organise the *visualisation* of the above information. Further descriptions of the categories encompassing each dimension are given within table III, coded as $[Xi]$, where i is the category identifier within a dimension, X, where $X \in \{A,B,C,D,E\}$.

We expand these five dimensions into 34 categories (sub-dimensions) and table III provides a weighting of these categories in the rightmost column as calculated as a percentage of the number of primary studies per category divided by the total of primary studies. The top five weighted categories (bolded in table III) highlight what most studies recommend documenting in API documentation, with the top three falling under the Usage Description dimension.

The majority (71%) of studies advocate for **code snippets** as a necessary piece in the API documentation puzzle [A5]. While code snippets generally only reflect small portions of API functionality (limited to 15–30 LoC), this is complimented by **step-by-step tutorials** (57% of studies) that tie in multiple (disparate) components of API functionality, generally with some form of screenshots, demonstrating the development of a non-trivial application using the API step-by-step [A6]. The third highest category weighted was also under the Usage Description dimension, being **low-level reference documentation** at 52% [A2]. These three categories were the only categories to be weighted as majority categories (i.e., their weighting was above 50%). The fourth and fifth highest weights are **an entry-level purpose/overview of the API** (48%) that gives a brief motivation as to why a developer should choose a particular API over another [B1] and **consistency in the look and feel** of the documentation throughout all of the API's official documentation (43%) [E6].

V. Threats to Validity

Threats to *internal validity* concern factors internal to our study that may affect results. Guidelines on producing systematic reviews [5] suggest that single researchers conducting

TABLE III
AN OVERVIEW OF THE 5 DIMENSIONS AND CATEGORIES (SUB-DIMENSIONS) WITHIN OUR PROPOSED TAXONOMY.

Key	Description: Dimensions A=Usage Description; B=Design Rationale; C=Domain Concepts; D=Support Artefacts; E=Documentation Presentation	Primary Studies	Total (%)
[A1]	Quick-start guides to rapidly get started using the API in a specific programming language.	[S4, S9, S10]	3/21 (14%)
[A2]	**Low-level reference manual documenting all API components to review fine-grade detail.**	**[S1, S3, S4, S8, S9, S10, S11, S12, S15, S16, S17]**	**11/21 (52%)**
[A3]	Explanations of the API's high-level architecture to better understand intent and context.	[S1, S2, S4, S11, S14, S16, S19, S20]	8/21 (38%)
[A4]	Source code implementation and code comments (where applicable) to understand the API author's mindset.	[S1, S4, S7, S12, S13, S17, S20]	7/21 (33%)
[A5]	**Code snippets (with comments) of no more than 30 LoC to understand a basic component functionality within the API.**	**[S1, S2, S4, S5, S6, S7, S9, S10, S11, S14, S15, S16, S18, S20, S21]**	**15/21 (71%)**
[A6]	**Step-by-step tutorials, with screenshots to understand how to build a non-trivial piece of functionality with multiple components of the API.**	**[S1, S2, S4, S5, S7, S9, S10, S15, S16, S18, S20, S21]**	**12/21 (57%)**
[A7]	Downloadable source code of production-ready applications that use the API to understand implementation in a large-scale solution.	[S1, S2, S5, S9, S15]	5/21 (24%)
[A8]	Best-practices of implementation to assist with debugging and efficient use of the API.	[S1, S2, S4, S5, S7, S8, S9, S14]	8/21 (38%)
[A9]	An exhaustive list of all major components that exist within the API.	[S4, S16, S19]	3/21 (14%)
[A10]	Minimum system requirements and dependencies to use the API.	[S4, S7, S13, S17, S19]	5/21 (24%)
[A11]	Instructions to install or begin using the API and details on its release cycle and updating it.	[S4, S7, S8, S9, S11, S13, S16, S19]	8/21 (38%)
[A12]	Error definitions that describe how to address a specific problem.	[S1, S2, S4, S5, S9, S11, S13]	7/21 (33%)
[B1]	**A brief description of the purpose or overview of the API as a low barrier to entry.**	**[S1, S2, S4, S5, S6, S8, S10, S11, S15, S16]**	**10/21 (48%)**
[B2]	Descriptions of the types of applications the API can develop.	[S2, S4, S9, S11, S15, S18]	6/21 (29%)
[B3]	Descriptions of the types of users who should use the API.	[S4, S9]	2/21 (10%)
[B4]	Descriptions of the types of users who will use the product the API creates.	[S4]	1/21 (5%)
[B5]	Success stories about the API used in production.	[S4]	1/21 (5%)
[B6]	Documentation to compare similar APIs within the context to this API.	[S2, S6, S13, S18]	4/21 (19%)
[B7]	Limitations on what the API can and cannot provide.	[S4, S5, S8, S9, S14, S16]	6/21 (29%)
[C1]	Descriptions of the relationship between API components and domain concepts.	[S3, S10]	2/21 (10%)
[C2]	Definitions of domain-terminology and concepts, with synonyms if applicable.	[S2, S3, S4, S6, S7, S10, S14, S16]	8/21 (38%)
[C3]	Generalised documentation for non-technical audiences regarding the API and its domain.	[S4, S8, S16]	3/21 (14%)
[D1]	A list of FAQs.	[S4, S7]	2/21 (10%)
[D2]	Troubleshooting suggestions.	[S4, S8]	2/21 (10%)
[D3]	Diagrammatically representing API components using visual architectural representations.	[S6, S13, S20]	3/21 (14%)
[D4]	Contact information for technical support.	[S4, S8, S19]	3/21 (14%)
[D5]	A printed/printable resource for assistance.	[S4, S6, S7, S9, S16]	5/21 (24%)
[D6]	Licensing information.	[S7]	1/21 (5%)
[E1]	Searchable knowledge base.	[S3, S4, S6, S10, S14, S17, S18]	7/21 (33%)
[E2]	Context-specific discussion forum.	[S4, S10, S11]	3/21 (14%)
[E3]	Quick-links to other relevant documentation frequently viewed by developers.	[S6, S16, S20]	3/21 (14%)
[E4]	Structured navigational style (e.g., breadcrumbs).	[S6, S10, S20]	3/21 (14%)
[E5]	Visualised map of navigational paths to certain API components in the website.	[S6, S14, S20]	3/21 (14%)
[E6]	**Consistent look and feel of documentation.**	**[S1, S2, S3, S5, S6, S8, S10, S15, S20]**	**9/21 (43%)**

their reviews should discuss the review protocol, inclusion decisions, and data extraction with a third party. In this paper, we have presented the early outcomes of our systematic review, which has utilised the test-retest methodology as a measure of reliability. MacDonell et al. [15] states that a defining characteristic of any SMS is to test the reliability of the review and extraction processes. We plan to mitigate this threat by conducting *inter*-relater reliability with the continuation of this work, using independent analysis and conflict resolution as per guidelines suggested by Garousi and Felderer [16]. Similarly, the development of our taxonomy would benefit from an inter-rater reliability categorisation of a sample of papers to both ensure that our weightings of categories are reliable and that the categories and dimensions fit the objectives of the taxonomy. Furthermore, a future user study (see section VI) will be needed to assess whether the extracted information from API documentation actually impacts on developer productivity, and the usefulness of such a taxonomy should be evaluated.

Threats to *external validity* represent the generalisation of the observations we have found in this study. While we have used a broad range of literature that encompasses API documentation guidelines, we acknowledge that not all papers contributing to API documentation may have been captured

in the taxonomy. All efforts were made to include as many papers as possible given our filtering technique, though it is likely that some papers filtered out (e.g., papers not in English) may alter our conclusions, introducing conflicting recommendations. However, given the consistency of these trends within the studies that were sourced, we consider this a low likelihood.

Threats to *construct validity* relates to the degree by which the data extrapolated in this study sufficiently measures its intended goals. Automatic searching was conducted in the SMS by choice of three popular databases (see section III-A). As a consequence of selecting multiple databases, duplicates were returned. This was mitigated by manually curating out all duplicate results from the set of studies returned. Additionally, we acknowledge that the lack of manual searching of papers within particular venues may be an additional threat due to the misalignment of search query keywords to intended papers of inclusion. Thus, our conclusions are only applicable to the information we were able to extract and summarise, given the primary sources selected.

VI. Conclusions & Future Work

API documentation is an aspect of quality of software, as it facilitates the developer's productivity and assists with evolution. Improving the quality of the documentation of third party APIs improves the quality of applications built using them. To date, we did not find a systematic literature review that offers a consolidated taxonomy of key recommendations to improve API documentation. Moreover, there has been little work on mapping the research produced in this space against the techniques used to arrive at the recommendations.

Starting with 4,501 papers potentially relating to API documentation, we identified 21 key relevant studies, and synthesised a taxonomy of the various documentation aspects that should improve API documentation quality. Furthermore, we also capture the most commonly used analysis techniques used in the academic literature. Figure 1 highlights that a majority of these studies employ interviews and questionnaires, and only some undertake structured documentation analysis.

In future revisions of this work, we intend to use our results as the input to a restricted systematic literature review in API documentation artefacts. In doing so, we will consider conducting the following:

- improve reliability metrics of our study (see section V) with an inter-rater reliability method;
- assess our taxonomy using a user study involving industry developers to assess the applicability and efficacy of these recommendations—this will empirically reflect what is important from a *practitioner* point of view;
- evaluate the selected studies to assess the effectiveness of the various approaches used in the conclusions;
- conduct a heuristic validation of the taxonomy against intelligent APIs, given the current trend in SE that is exploring how machine learning and artificial intelligence-based applications may affect existing approaches;
- arrive at a relevance ranking for each of the 34 categories, based on further validation, user studies and the current weights.

We believe the results of this preliminary empirical work may provide further insight for future follow-up user studies with developers. Whilst our aim is to eventually improve the quality of API documentation, the ultimate goal is improving the developer's experience when producing systems and, therefore, improving the efficacy and productivity at which software is produced within industry. We hope that API designers will utilise the taxonomy produced in this paper as a weighted checklist for what should be considered in their own APIs.

References

[1] L. McLeod and S. G. MacDonell, "Factors that affect software systems development project outcomes: A survey of research," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 24, 2011.

[2] K. Nybom, A. Ashraf, and I. Porres, "A Systematic Mapping Study on API Documentation Generation Approaches," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Prague, Czech Republic, pp. 462–469.

[3] R. B. Watson, "Development and application of a heuristic to assess trends in API documentation," in *30th ACM international conference on Design of communication*. Seattle, Washington, USA: ACM, 2012, pp. 295–302.

[4] 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.

[5] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Tech. Rep., 2007.

[6] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering." in *EASE*, 2008, pp. 68–77.

[7] M. Usman, R. Britto, J. Börstler, and E. Mendes, "Taxonomies in software engineering: A Systematic mapping study and a revised taxonomy development method," *Information and Software Technology*, vol. 85, pp. 43–59, May 2017.

[8] R. L. Glass, I. Vessey, and V. Ramesh, "Research in software engineering: an analysis of the literature," *Information and Software Technology*, vol. 44, no. 8, pp. 491–506, 2002.

[9] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Information and Software Technology*, vol. 106, pp. 101 – 121, 2019.

[10] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," 1990.

[11] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, no. 4, pp. 571–583, Apr. 2007.

[12] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–17, Mar. 1977.

[13] R. J. Wieringa and J. M. Heerkens, "The methodological soundness of requirements engineering papers: a conceptual framework and two case studies," *Requirements engineering*, vol. 11, no. 4, pp. 295–307, 2006.

[14] J. Singer, S. E. Sim, and T. C. Lethbridge, "Software engineering data collection for field studies," in *Guide to Advanced Empirical Software Engineering*. Springer Science & Business Media, Nov. 2007, pp. 9–34.

[15] S. MacDonell, M. Shepperd, B. Kitchenham, and E. Mendes, "How reliable are systematic reviews in empirical software engineering?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 676–687, Sep. 2010.

[16] V. Garousi and M. Felderer, "Experience-based guidelines for effective and efficient data extraction in systematic reviews in software engineering," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE'17. New York, NY, USA: ACM, 2017, pp. 170–179.