Requirements of API Documentation: A Case Study into Computer Vision Services

Alex Cummaudo, Rajesh Vasa, John Grundy, and Mohamed Abdelrazek

Abstract—Using cloud-based computer vision services is gaining traction, where developers access AI-powered components through familiar RESTful APIs, not needing to orchestrate large training and inference infrastructures or curate/label training datasets. However, while these APIs *seem* familiar to use, their non-deterministic run-time behaviour and evolution is not adequately communicated to developers. Therefore, improving these services' API documentation is paramount—more extensive documentation facilitates the development process of intelligent software. In a prior study, we extracted 34 API documentation artefacts from 21 seminal works, devising a taxonomy of five key requirements to produce quality API documentation. We extend this study in two ways. Firstly, by surveying 104 developers of varying experience to understand what API documentation artefacts are of *most value* to practitioners. Secondly, identifying which of these highly-valued artefacts are or are not well-documented through a case study in the emerging computer vision service domain. We identify: (i) several gaps in the software engineering literature, where aspects of API documentation understanding is/is not extensively investigated; and (ii) where industry vendors (in contrast) document artefacts to better serve their end-developers. We provide a set of recommendations to enhance intelligent software documentation for both vendors and the wider research community.

Index Terms—Intelligent Web Services and Semantic Web, Code Documentation, Computer Vision

1 Introduction

Improving API documentation quality is a valuable task for any API. Succinct API documentation of good quality facilitates productivity [32, 38, 37], and therefore improved quality is better engineered into a system [35]. Where application developers integrate new services into their systems via APIs, 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"*). As a real-world use case, consider intelligent computer vision services, in which an AI-based component produces a non-deterministic result based on a machine-learnt data-driven algorithm, rather than a predictable, rule-driven one [10]. These services use machine intelligence to make predictions on images such as object labelling or facial recognition [W1–15]. The impacts of poor and incomplete documentation results in developer complaints on online discussion forums such as Stack Overflow [11]. Many comments show that developers do not think in the non-deterministic mental model of the designers who created the computer vision services. They ask many varied questions from their peers to try and clarify their understanding.

It is therefore important to ensure developers have access to high-quality API documentation artefacts when consuming these services. Vendors should cover all documentation artefacts that the wider developer community find valuable, and the research community should aide in this process by investigating with types of information that comprise these artefacts, or the aspects of information design to best present this information. What causes a developer to be confused when using an API, and how to mitigate it via improved documentation, has been largely explored by researchers for *conventional* APIs (an overview is provided in section 2). Various studies provide a myriad of recommendations into the value of API documentation artefacts based on both qualitative and quantitative analyses, involving developer opinions (from surveys), observation of developers, event logging or content analysis (see fig. 3). Such guidelines propose ways for developers, managers, and solution architects can construct systems better with improved documentation.

However, there does not yet exist a consolidated *systematic* review of this literature. Further, few studies offer a taxonomy to consolidate these guidelines together, and there still lacks a consolidated effort to capture guidelines on the requirements of good quality API documentation. Studies that produce these guidelines from literature are largely scattered across multiple sources. Investigating the ways by which these guidelines are produced can provide software engineering researchers with better insight into the research methods and data collection techniques used to produce these guidelines. Some studies, for example, use case studies, others use focus groups and brainstorming, or interviews and surveys. The extent to which researchers rely on developer opinion for API documentation guidelines is evident, and gaps in the methodological approaches that researchers use should be emphasised to shine light into new ways of conducting research in this important area. Furthermore, systematically capturing the information distilled

- *A. Cummaudo and R. Vasa are with the Applied Artificial Intelligence Institute, Deakin University, 1 Gheringhap St, Geelong Victoria 3220, Australia. E-mail: {ca, rajesh.vasa}@deakin.edu.au.*
- *J. Grundy is with the Faculty of Information Technology, Monash University, Wellington Rd, Clayton Victoria 3800, Australia. E-mail: john.grundy@monash.edu.*
- *M. Abdelrazek is with the School of Information Technology, Deakin University, 1 Gheringhap St, Geelong Victoria 3220, Australia. E-mail: mohamed.abdelrazek@deakin.edu.au.*

from these guidelines into a readily accessible, consolidated taxonomy (designed to assist writing API documentation) must be validated in real-world circumstances to assess its efficacy with practitioners.

In our prior work, we proposed an API documentation taxonomy that was comprised of 21 key primary sources [9]. This paper significantly extends our previous work by addressing limitations in the existing taxonomy, thus refining it. Previously, we developed a metric for each dimension (topmost-layers) and category (leaf nodes) within the taxonomy [9]. This metric is an indication of the specific areas of API documentation software engineering researchers have focused their efforts, as measured by the ratio of papers that investigated or reported various issues concerning the documentation artefacts defined within our taxonomy. For the context of this paper, we refer to this metric as an 'in-literature' score, or ILS. Within this paper, we build upon this facet but *in-practice* by assessing the efficacy of our taxonomy against developers using a survey instrument inspired by the System Usability Scale (SUS) [6]. Each artefact within the taxonomy is measured against this instrument for its utility, and a metric is produced to indicate how well developers *value* each of these artefacts. We refer to this metric as an 'in-practice' score, or IPS. (Details for how the IPS is calculated are in given in section 5.1.4.) We then identify the artefacts that are highly researched, the ones that developers demand the most, and where gaps in these artefacts remain for future research exploration.

Lastly, while our prior work focused on *generalised* API documentation, in this extension, we apply our taxonomy to a case study of interest: i.e., better documenting computer vision services. We empirically assess the taxonomy against three popular computer vision services, namely Google Cloud Vision [W1], Amazon Rekognition [W2] and Azure Computer Vision [W3]. For each category in our taxonomy, we assess whether the respective service's documentation contains, partially-contains or does not contain the documentation artefact from our taxonomy, thus determining the extent to which the requirements of good API documentation are met within the vendors' own documentation. From this, we triangulate each ILS and IPS value against the service's level of inclusion of its respective documentation artefact, thereby making a judgement as to where the services can improve their documentation to make them more complete. Lastly, we present a ranking of each artefact for where research or vendors should be focus their documentation efforts that is of high value to both developers *and* to industry vendors.

Thus, through this triangulation of the taxonomy with existing literature, utility to practitioners, and application via a case study (computer vision services), we summarise three aspects of API documentation by identifying:

(i) the documentation artefacts that been extensively studied by researchers, and those that warrant further attention by the software engineering research community (via high/low ILS values);
(ii) the documentation artefacts that are considered to be the most- and least-important from a practitioner's point of view (via high/low IPS values);
(iii) the documentation artefacts that have been well-established by vendors (via our case study on three prominent computer vision services).

To demonstrate how our taxonomy was developed, we include an extended revision of the systematic mapping study (SMS) from our existing work. The taxonomy we proposed consists of five key requirements: (1) Descriptions of API Usage; (2) Descriptions of Design Rationale; (3) Descriptions of Domain Concepts; (4) Existence of Support Artefacts; and (5) Overall Presentation of Documentation. Following this, we developed a survey instrument to assess the overall utility of each of the artefacts that contribute towards these five requirements, which consisted of 43 questions of alternating positive and negative sentiment. We then narrow our focus down to our case study by applying the prioritised documentation artefacts (as identified by the survey) to three computer vision services. Once our surveys were complete, we provide some general guidelines as to where cloud computer vision services can make improvements to their API documentation. Lastly, we compare and contrast the results from our SMS to the results of the survey and of our case study, thereby identifying where future research efforts into API documentation should focus to give the biggest value back to practitioners.

Our key contributions in this work are:

- a score metric for each category that indicates where the highest research priorities have been in the existing literature;
- a score metric assessing the efficacy of the 34 categories that empirically reflects what artefacts are of the highest value from a *practitioner* point of view;
- a heuristic validation of each artefact against computer vision services, assessing where existing computer vision service API documentation needs improvement;
- a number of practical recommendations for computer vision service vendors to better improve the quality of their API documentation; and
- an identification of the gaps for future research into API documentation based on the highest need by developers but, so far, has captured the least attention by researchers.

This paper is structured as follows: section 2 presents related work; section 3 is divided into two subsections, the first describing how primary sources were selected in the SMS with the second describing the development of our taxonomy from these sources; section 4 presents the taxonomy; section 5 describes how we developed a survey instrument of 43 questions to validate the taxonomy against developers, and assess its efficacy against the three popular computer vision services selected; section 6 presents the findings from our validation analysis; section 7 describes the threats to validity of this work; and section 8 provides concluding remarks and the future directions of this study. Additional materials are provided in appendixes A–F; referenced online artefacts are prefixed with 'W' and can be found in appendix C.

2 Related Work

2.1 Systematic Reviews in Software Documentation

Systematic reviews into how developers produce and use software documentation gives researchers consolidated insights into the efforts of multiple, disparate API documentation studies. For example, a recent 2018 study explored 36 API documentation generation tools and approaches, and analysed the tools developed and their inputs and documentation outputs [39]. The findings from this study emphasise that the largest effort in API documentation tooling is to assist developers to generate either example code snippets and/or templates or natural language descriptions of the API directly from the program's source code. These snippets or descriptions can then be placed in the API documentation, thereby increasing the efficiency at which API documentation can be written. Additionally, tools from 12 studies target the maintainability of existing APIs of existing APIs, while tools from 11 studies target the correctness and accuracy of the documentation by validating that what is written in the documentation is accurate to the technical structure of the API. From the end-developer's perspective, some tools (17 studies) help target improvements to the developer's understandability and learnability of new APIs by linking in examples directly with questions such as on Stack Overflow. However, the results from this study regards the *tooling* used to either assist in producing, validating or learning from API documentation. While this is a systematic study with key insights into the types of tooling produced, there is still a gap for a SMS in what *guidelines* have been produced by the literature in developing natural language documentation itself—and how well developers *agree* to those guidelines—which our work has addressed.

An extensive SMS into studies presented in the *overall* software documentation domain was given in Zhi et al. [59]. This study reviewed a set of 69 papers from 1971 to 2011 to develop a systematic map on the various research aspects relating to documentation cost, benefit and quality, finding that 38% of papers propose novel techniques while 29% contribute empirical evidence (i.e., validation and evaluation papers—see section 3.1.4). The authors find that a majority of papers discuss quality aspects of software documentation, namely the quality attributes of completeness, consistency and accessibility, and that the main usage of software documentation regards maintenance aid and program comprehension. Another key insight—relevant to our study—found that, on average, survey-based studies into documentation involved 106 participants and generally these participants were from the same (or only two) organisations. However, unlike our study, this study formalises the documentation efforts of *any* software document, and not exclusively into API documentation artefacts required to help developers produce software. Further, our study differs in that the results from our study are consolidated into a structured taxonomy, instead of a meta-model which Zhi et al. perform, which is then triangulated against a real-world use case (i.e., intelligent computer vision services) and software developers via a survey.

2.2 API Usability and Documentation Knowledge

API usability and its impact on documentation knowledge is an imperative area of study, since it provides useful links between API documentation and more technical issues related to API design or tools. Extensive discussions from Myers and Stylos [38] and Myers et al. [37] encapsulate a 30-year effort to evaluate and improve API usability through lenses adapted human-computer interaction research. Essentially, by treating a developer as the 'end-user' of an API (i.e., interacting and programming with the API in their own systems), the authors discuss various case studies by which API usability was improved by various human-centred approaches, resulting in improved learnability of the API in addition to improved productivity and effectiveness in using the API. While the methods are primarily used for end-user usability testing, their observations highlight the importance of good aesthetic and interaction design of developer's tooling and the need for new tooling to augment what developers already do to reduce learning overhead. An extensive review of the usability methods used, and their benefits to API usability, demonstrates how various techniques—grounded through established usability guidelines and frameworks—can be used to assess how an API's usability impacts its key stakeholders (i.e., API designers, developers, and end-users). The role of API *documentation* in context to an API's overall usability is imperative; for instance, limited documentation on a particular API (and limited code snippets) is often a key complaint to poor API usability [38]. Exploring aspects on information design elements within API documentation is therefore critical to mitigate such complaints.

In Watson [56], the authors performed a heuristic assessment from 35 popular APIs against 11 high-level universal design elements of API documentation. Of these 35 APIs, 28 were open-source software repositories and seven came from commercial independent software vendors. Two coders manually inspected each API's respective documentation sets, starting from the documentation's entry page and using the navigation features of the documentation to further explore the documentation. Both coders evaluated each of the 11 heuristics, noting whether they could be found. This study highlighted how many APIs, even popular ones, fail to grasp these basic design elements. For example, 25% of the documentation sets did not provide any basic overview documentation to the API. Therefore, from a practitioner's perspective, the study describes a high-level overview of how certain documentation artefacts address their needs and whether they are typically found in documentation. However, while the methodological approach used in this study to assess the heuristics is similar to our approach, the heuristics themselves used within Watson's study is based on only three seminal works and only contains 11 design elements. Our study extends these heuristics and structures them into a consolidated, hierarchical taxonomy which we then validate against practitioners.

A taxonomy of distinct knowledge patterns within reference documentation by Maalej and Robillard [33] classified 12 distinct knowledge types. Unlike our work, which uses a SMS of existing studies as the source of our taxonomy development, this study uses a grounded method via theoretical sampling of the API documentation of two mature (extensively documented) open source systems. This was performed by each author to elicit a list of knowledge types over an iterative six month process. The taxonomy was then evaluated against the JDK 6 and .NET 4.0 frameworks using a sample of 5574 documentation units and 17 trained coders to assign each knowledge type to the documentation unit.

Results showed that the functionality and structure of these APIs are well-communicated, although core concepts and rationale about the API are quite rarer to see. The authors also identified low-value 'non-information'—described as documentation that provides uninformative boilerplate text with no insight into the API at all—which was substantially present in the documentation of methods and fields in the two frameworks. They recommend that developers factor their 12 distinct knowledge types into the process of code documentation, thereby preventing low-value non-information, and thus developers can use the patterns of knowledge to evaluate the content, organisation, and utility of their own 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 utilised and the source of information of our taxonomy (i.e., existing literature).

2.3 Computer Vision Services

Recent studies into cloud-based computer vision services have demonstrated that poor reliability and robustness in computer vision can 'leak' into end-applications if such aspects are not sufficiently appreciated by developers. A study by Hosseini et al. [21] showed that Google Cloud Vision's labelling fails when as little as 10% noise is added to the image. Facial recognition classifiers are easily confused by modifying pixels of a face and using transfer learning to adapt one person's face into another [55]. Our own prior work found that the non-deterministic evolution of these types of services is not adequately communicated to developers [10], resulting in lost developer productivity whereby developers ask fundamental questions about the concepts behind these services, how they work, and where better documentation can be found [11]. This paper continues this line of research by providing a means for service providers to better document their services using a taxonomy and suggested improvements.

3 Taxonomy Development

We developed our taxonomy under two primary phases. First, we conducted a SMS identifying API documentation studies, following guidelines by Kitchenham and Charters [26] and Petersen et al. [42] (section 3.1). A high level overview of this first phase is given in fig. 2. Second, we followed a software engineering taxonomy development method by Usman et al. [54] (section 3.2) based on the findings of our SMS, which involved an extensive validation involving real-world developers and contextualised with computer vision APIs (section 5).

3.1 Systematic Mapping Study

3.1.1 Research Questions (RQs)

The first step in producing our SMS was to pose two RQs:

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

Our intent behind RQ1 was to collect as many studies provided by literature on how API documentation should be written using natural language, i.e., not using assistive tooling. In this regard, documentation 'knowledge' encompasses any natural language API documentation artefact associated with the implementation of an application using a third-party API. As the goals of this study are to arrive at a taxonomy encapsulating the requirements of good API documentation (section 4), we sought to arrive at studies that provide useful information to developers that informs the relevance and value of which aspects of API documentation are more useful than others. This captures the knowledge that developers need to know about what aspects of their APIs should be documented and the artefacts by which they do this. This helped us shape and form the taxonomy provided in section 4. Secondly, RQ2's intent was to understand how the studies derive at their conclusions, thereby helping us identify gaps in literature where future studies can potentially focus.

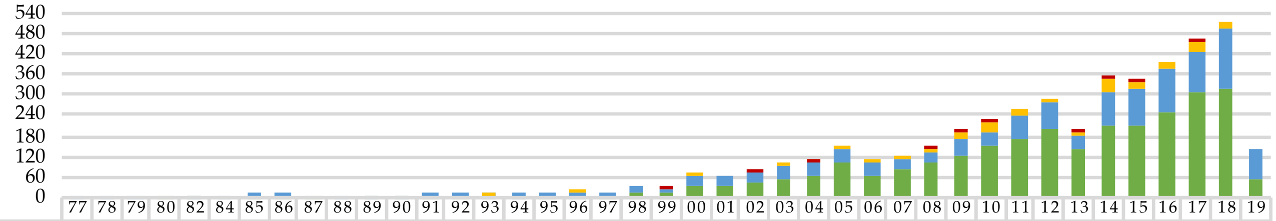
3.1.2 Automatic Filtering

As done in similar software engineering studies [17, 54, 15], we explored automatic filtering of online databases. We defined which SWEBOK knowledge areas [22] were relevant to devise a search query. Our search query was built using related knowledge areas, 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)

We executed the query on all available metadata (title, abstract and keywords) in May 2019 against Web of Science[1] (WoS), Compendex/Inspec[2] (C/I) and Scopus[3]. We selected three particular primary sources given their relevance in software engineering literature (containing the IEEE, ACM, Springer and Elsevier databases) and their ability to support advanced queries [5, 26]. A total 4,501 results[4] were found, with 549 being duplicates. Table 1 displays our results in further detail (duplicates not omitted); fig. 1 shows an exponential trend of API documentation publications produced within the last two decades. (As this search was conducted in May 2019, results taper in 2019.)

[1]http://apps.webofknowledge.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.



	77	78	79	80	82	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
🟥 Other	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	2	1	0	0	1	1	1	0	1	0	0
🟨 Book (Chapter)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3	0	0	0	1	0	0	4	1	11	12	11	17	15	21	22	13	10	38	19	21	30	14	0
🟦 Journal Article	2	1	1	0	2	2	5	3	0	2	6	5	3	7	5	11	13	14	7	17	18	25	26	26	37	45	37	37	29	27	50	46	67	78	34	102	108	121	123	176	86
🟩 Conference Paper	0	0	0	1	1	0	4	4	1	0	1	1	3	6	7	2	7	7	12	19	13	36	36	47	59	62	103	65	83	103	126	150	172	201	145	208	208	250	303	320	55

Fig. 1: Search results by year and venue type.

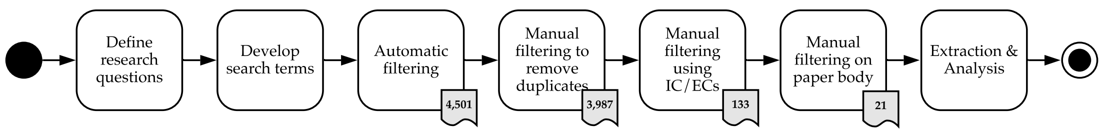

Define research questions
Develop search terms
Automatic filtering
4,501
Manual filtering to remove duplicates
3,987
Manual filtering using IC/ECs
133
Manual filtering on paper body
21
Extraction & Analysis

Fig. 2: A high level overview of the filtering steps from defining and executing our search query to the data extraction of our primary studies. Number of accepted papers resulting from each filtering step is shown.

TABLE 1: 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

3.1.3 *Manual Filtering*

A follow-up manual filtering stage followed the 4,501 results obtained by automatic filtering. As described below, we applied the following inclusion criteria (IC) and exclusion criteria (EC) to each result:

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 discuss artefacts that document APIs;
IC3 Studies must be relevant to software engineering 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.

Each of these ICs and ECs were applied to every paper after exporting all metadata of our results to a spreadsheet. The first author then curated the publications using the following revision process.

Firstly, we read the publication source—to rapidly omit non-software engineering papers—as well as the author keywords, title, and abstract of all 4,501 studies. As some studies were duplicated between our three primary sources, we needed to remove any repetitions. We sorted and reviewed any duplicate DOIs and fuzzy-matched all very similar titles (i.e., changes due to punctuation between primary sources), thereby retaining only one copy of the paper from a single database. Similarly, as there was no limit do our date ranges, some studies were republished in various venues (i.e., same title but different DOIs). These were also removed using fuzzy-matching on the title, and the first instance of the paper's publication was retained. This second phase resulted in 3,987 papers.

Secondly, we applied our inclusion and exclusion criteria to each of the 3,987 papers by reading the abstract. Where there was any doubt in applying the criteria to the abstract alone, we automatically shortlisted the study. We rejected 427 studies that were unrelated to software engineering, 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. This resulted in 133 studies being shortlisted to the final phase.

Thirdly, we re-evaluated each shortlisted paper by re-reading the abstract, the introduction and conclusion. We removed a further 64 studies that were on API usability or non API-related documentation (i.e., code commenting). At this stage, we decided to refine our exclusion criteria to better match the research goals of this study by including the word 'natural language' documentation in EC2. This removed studies where the focus was to improve technical documentation of APIs such as data types and communication schemas. Additionally, we removed 26 studies as they were related to introducing new tools (EC3), 3 were focused on tools to mine API documentation, 7 studies where no guidelines 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 final stage resulted in 21 primary studies for further analysis, and the mapping of primary study identifiers to references S1–21 can be found in appendix D.

As a final phase, we conducted reliability analysis of our shortlisting method. We conducted intra-rater reliability of our 133 shortlisted papers using the test-retest approach suggested by Kitchenham and Charters [26]. We re-evaluated a random sample of 10% of the 133 shortlisted papers a week after initial studies were shortlisted. This resulted in *substantial agreement* [31], measured using Cohen's kappa ($\kappa = 0.7547$).

3.1.4 Data Extraction & Systematic Mapping

Of the 21 primary studies, we conducted abstract keywording adhering to Petersen et al.'s guidelines [42] 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 used in the requirements engineering field by Wieringa et al. [58]. These are: *evaluation papers*, which evaluates existing techniques currently used in-practice; *validation papers*, which investigates proposed techniques not yet implemented in-practice; *experience papers*, which are written by practitioners in the field and provide insight into their experiences of adopting existing techniques; and *philosophical papers*, which presents new conceptual frameworks that describes a language by which we can describes our observations of existing or new techniques, thereby implying a new viewpoint for understanding phenomena. For example, documenting APIs using code snippets is a commonly used practice by developers (see the primary sources listed in appendix A), and conducting an experiment exploring how quickly practitioners achieve this would be an evaluation paper. In contrast, a validation paper explores novel techniques that are proposed but not yet implemented in practice; for example, a paper proposing that APIs should document success stories so that developers know where, why, and how the API was successfully implemented may test this novel technique via field study experiments (e.g., interviewing developers on the new technique) without reference to real-world examples. A paper written by a group of developers sharing their insights into the improvements of their documentation before and after providing extensive tutorials would be an experience paper. Philosophical papers may propose entirely new vocabulary to explore API documentation, devising new frameworks from which other researchers can explore the field from a new viewpoint.

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 software engineering field study techniques [51]. 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 2: Data extraction form

Data item(s)	Description
Citation metadata	Title, author(s), years, publication venue, publication type
Artefact(s) discussed	As per IC2, the study must identify at least one API documentation artefact
Evaluation method	Did the authors evaluate their proposed artefacts? If so, how?
Primary technique	The primary technique used to devise the artefact(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 et al.'s taxonomy

Table 2 describes our data extraction form, which was used to collect relevant data from each paper. Figure 3 presents our systematic mapping, where each study is mapped to one (or more, if applicable) of methodologies plotted against Wieringa et al.'s research approaches. We find that a majority of these studies survey developers using direct techniques (i.e., interviews and questionnaires) and some performing structured documentation analysis. Few studies report recent experiences; literature reports the artefacts that document APIs from evaluation research, in addition to some validation studies. There are few experience papers describing anecdotal evidence, and almost no philosophical papers that describe new conceptual ways at approaching API documentation as a large majority of existing work either evaluates existing (in-practice) strategies or validates the effectiveness of new strategies.

3.2 Development of the Taxonomy

A majority of taxonomies produced in software engineering studies are often made extemporaneously [54]. For this reason, we decided to proceed with a systematic approach to develop our taxonomy using the guidelines provided by Usman et al. [54], which are extended from lessons learned in more mature domains. In this subsection, we outline the 4 phases and 13 steps taken to develop our taxonomy based on Usman et al.'s technique. Usman et al.'s final *validation* phase is largely detailed within section 5 after we present our taxonomy in section 4.

Formally, Usman et al. provides guidelines to define these units under the first six stages under the planning phase. In our study, our preliminary phase involves answering the following:

(1) *define the software engineering knowledge area*: The software engineering knowledge area, as defined by the SWEBOK, is software construction;

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

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

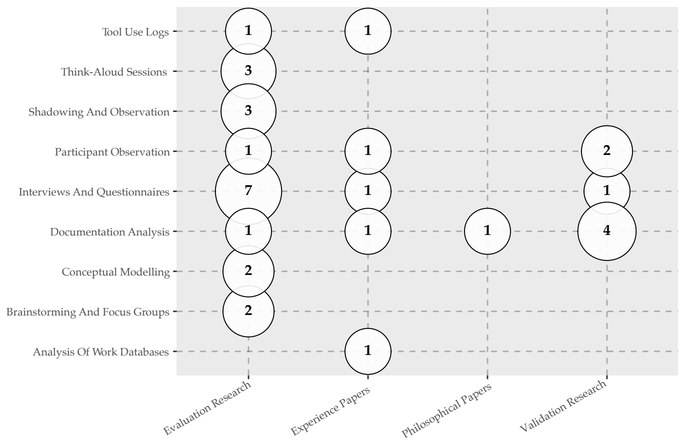


Fig. 3: Systematic map: field study technique vs research type

(4) *define the classification structure*: The classification structure of our proposed taxonomy is *hierarchical*;
(5) *define the classification procedure*: The procedure used to classify the documentation artefacts is qualitative;
(6) *define the data sources*: The basis of the taxonomy is derived from field study techniques (see section 3.1.4).

3.2.1 Identification and extraction phase

The second phase of the taxonomy development involves **(7)** *extracting all terms and concepts* from relevant literature, which we have achieved from our SMS. These terms are then consolidated by **(8)** *performing terminology control*, as some terms may refer to different concepts and vice-versa. For example, Watson defines one of the heuristics used in the study's experiment as "sample apps to understand how to use the elements of an API in context and as another source from which to copy program code... a sample app is a complete application that includes examples of the API as well as the other functions that comprise a complete program" [56]. In this case, the term 'sample app', 'program code', and 'complete application' were extracted as a term of interest and noted. Similarly, in Robillard [45], the phrase 'applications' is used to define a category of example code snippets which "consists of code segments from complete applications" and is generally some form of "demonstration samples sometimes distributed with an API... that developers can download from various source code repositories" [45]. Again, the phrase 'complete applications', 'demonstration samples', 'download', and 'source code' was identified as a terms of interest and noted. Once all papers were read, we consolidated a list of all of these noted highlights to help consolidate the terms and perform terminology control. In this example, the phrase 'Downloadable source code demonstrating complete sample applications' was consolidated from both Watson and Robillard's studies, which—in addition to the other primary studies that iteratively changed wording slightly due to steps (9–10)—formed the basis of the taxonomy dimension [**A7**].

3.2.2 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 each dimension, i.e., extracting the categories first and then nominating which dimensions these categories fit into using an iterative approach. As we used a bottom-up approach, 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*. The taxonomy is as simple as a checklist that can be heuristically applied to API documentation, and each dimension is malleable and covers a broad spectrum of artefacts; while we do not anticipate any further dimensions to be added, new categories can easily be fitted into one of the dimensions (see section 8). We provide guidelines for use in our application of the taxonomy against computer vision services within sections 4 and 6.

3.2.3 Validation phase

In the final phase of taxonomy development, taxonomy designers must **(13)** *validate the taxonomy* to assess its usefulness. Usman et al. [54] describe three approaches to validate taxonomies: (i) orthogonal demonstration, in which the taxonomy's orthogonality is demonstrated against the dimensions and categories, (ii) benchmarking the taxonomy against similar classification schemes, or (iii) utility demonstration by applying the taxonomy heuristically against subject-matter examples. In our study, we adopt utility demonstration by use of a survey and heuristic application of the taxonomy against real-world case-studies (i.e., within the domain of computer vision services). This is is discussed in greater detail within section 5.

4 A Taxonomy for API Documentation

Our taxonomy consists of five dimensions (labelled A–E). These five dimensions are made of 34 categories, which represent API documentation artefacts that contribute towards these dimensions. In the context of our taxonomy, a category can represent (i) discrete and self-contained documentation artefacts (e.g., quick start guides [**A1**]), (ii) additional information used to describe the API (e.g., licensing information about the API [**D6**]), or (iii) aspects regarding the information design of this documentation (e.g., consistent look and feel [**E6**]). **Collectively, the categories form the *requirements* of good quality API documentation, as expressed through the five dimensions.** When worded as questions, each dimension respectively covers the following:

- [**A**] **Descriptions of API Usage**: *how* does the developer use this API for their intended use case?
- [**B**] **Descriptions of Design Rationale**: *when* should the developer choose this particular API for their intended use case?
- [**C**] **Descriptions of Domain Concepts**: *why* does the developer select this particular API for their application's domain and does the API's domain align with the application's domain?

[A] Descriptions of API Usage

- [A1] Quick-start guides
- [A2] Low-level reference manual ✓
- [A3] Explanation of high level architecture ✓
- [A4] Introspection source code comments ✗
- [A5] Code snippets of basic component function ✓
- [A6] Step-by-step tutorials with multiple components
- [A7] Downloadable production-ready source code
- [A8] Best-practices of implementation
- [A9] An exhaustive list of all components
- [A10] Minimum system requirements to use the API
- [A11] Instructions to install/update the API and its release cycle
- [A12] Error definitions describing how to address problems

[B] Descriptions of the API's Design Rationale

- [B1] Entry-point purpose of the API ✓
- [B2] What the API can develop
- [B3] Who should use the API
- [B4] Who will use the applications built using the API ✗
- [B5] Success stories on the API
- [B6] Documentation comparing similar APIs to this API
- [B7] Limitations on what the API can/cannot provide

[C] Descriptions of Domain Concepts behind the API

- [C1] Relationship between API components and domain concepts
- [C2] Definitions of domain terminology
- [C3] Documentation for nontechnical audiences ✓

[D] Existence of Support Artefacts

- [D1] FAQs ✓
- [D2] Troubleshooting hints ✗
- [D3] API diagrams
- [D4] Contact for technical support ✓
- [D5] Printed guide
- [D6] Licensing information

[E] Overall Presentation of Documentation

- [E1] Searchable knowledge base ✓
- [E2] Context-specific discussion forums
- [E3] Quick-links to other relevant components ✗
- [E4] Structured navigation style ✓
- [E5] Visualised map of navigational paths ✗
- [E6] Consistent look and feel ✓

Fig. 4: Our proposed taxonomy: The requirements of good-quality API documentation (dimensions) represented through individual documentation artefacts (categories).

- **[D] Existence of Support Artefacts**: *what* additional API documentation can the developer find to aid their productivity?
- **[E] Overall Presentation of Documentation**: is the *visualisation* of the above information well organised and easy for the developer to digest?

Further descriptions of the categories encompassing each dimension are given within fig. 4 and appendix A, coded as $[Xi]$, where i is the category identifier within a dimension, X, where $X \in \{A, B, C, D, E\}$.

Appendix A shows which of the primary sources (S1–21) reports aspects of the artefacts described as an 'in-literature score' (ILS). This score is calculated as a percentage of the number of primary studies that investigated or reported various issues regarding the specific artefact divided by the total of primary studies (see section 6.1.2). This score is contrasted to the 'in-practice score' (IPS) which indicates the overall level of agreement that *practitioners* think such documentation artefacts are needed (see section 6.1.1). For comparative purposes, we illustrate a colour scale (from red to green) to indicate the relevancy weight between ILS and IPS values in appendix A as per their assigned, discretised intervals (see table 3). We also show illustrative interpretations of these generalised artefacts through italicised examples within appendix A. We then provide three columns that assesses the presence of these documentation artefacts against three popular computer vision services: Google Cloud Vision, AWS's Rekognition, and Azure Cloud Vision (abbreviated to GCV, AWS and ACV). A fully shaded circle (●) indicates that the documentation artefact was clearly found in the service, while a half-shaded circle (◑) indicates that the artefact was only partially present. An outlined circle (○) indicates that the service lacks the indicated documentation artefact within our taxonomy. This empirical assessment is further detailed in section 6.3, which outlines concrete areas in the respective services' documentation where improvements could be made, as well as hyperlinks to the documentation where relevant.

Figure 4 illustrates a condensed version of taxonomy. We provide iconography for the presence (✓) or non-presence (✗) of these artefacts in *all three* computer vision services assessed, per section 6.1.1.

5 Validating the Taxonomy

5.1 Survey Study

5.1.1 Designing the Survey

We followed the guidelines by Kitchenham and Pfleeger [27] on conducting personal opinion surveys in software engineering to validate our survey. In developing our survey instrument, we shaped questions around each of our 5 dimensions and 34 categories. To achieve this, we used Brooke's SUS [6] as a loose inspiration and re-shaped the 34 categories around a question that imitates the style of wording of questions used in the SUS. Each dimension was marked a numeric question (Q#3–7), and alphabetic sub-questions were marked for each sub-dimension or category.

We used closed questioning where respondents could choose an answer on a 5-point Likert-scale (1=*strongly disagree*, 2=*somewhat disagree*, 3=*neither agree nor disagree*, 4=*slightly agree* and 5=*strongly agree*). Like Brooke's study, each question alternated in positive and negative sentiment. Half of our questions were written where a likely common response would be in strong agreement and vice-versa for the other half, such that participants would have to "read each statement and make an effort to think whether they would agree or disagree with it" [6]. For example, the question regarding [**B7**] on API limitations was framed as: "*I believe it is important to know about what the limitations are on what the API can and cannot provide*" (Q4g), whereas the question regarding [**C1**] on domain concepts of the API was framed as: *"I wouldn't read through theory about the API's domain that relates theoretical concepts to API components and how both work together"* (Q5a).

In addition, the remaining eight questions asked demographical information. An extra open question asked for further comments. The full survey is provided in appendix F and anonymised survey data is available at https://bit.ly/33siqll.

5.1.2 Evaluating the Survey

After the first pass at designing questions was completed, we evaluated our survey on three researchers within our research group for general feedback. This resulted in minor changes, such as slight re-wording of questions and providing specific questions with examples (some with images). For example, the question regarding [**A9**] on an exhaustive list of all major components in the API was framed as *"I believe an exhaustive list of all major components in the API without excessive detail would be useful when learning an API"* (Q3i) with the example *"e.g., a computer vision web API might list object detection, object localisation, facial recognition, and facial comparison as its 4 components"*.

After this, we conducted reliability analysis using a test-retest approach on three developers within our group seven weeks apart. Using the R statistical computation environment [43], we conducted our analysis using the `irr` package [13] (as suggested in [18]) and resulted in an average intra-class correlation (ICC) of 0.63 which indicates a good overall index of agreement [8].

5.1.3 Recruiting Participants

Our target population for the study was application software developers with varying degrees of experience (including those who and who have not used computer vision services or related tools before) and varying understanding of fundamental machine learning concepts. We began by recruiting software developers within our research group using a group-wide message sent on our internal messaging system. Of the 44 developers in our group's engineering cohort,[5] 22 responses were returned, indicating an internal response rate of 50.00%. Based on the 22 results from this internal trial, we calculated the median time to our complete survey was just over 20 minutes.

For external participant recruiting, we shared the survey on social media platforms and online-discussion forums relevant to software development. We adopted a non-probabilistic snowballing sampling where the participants, at the end of the survey, were encouraged to share the survey link to others using *AddThis*.[6] Additionally, snowballing sampling was encouraged within members of our research group who were asked to share the survey. This sampling approach resulted in 38 external responses. A further 44 participants were recruited via Amazon Mechanical Turk[7]—often referred to as MTurk—which has been a successful approach adopted in previous software engineering surveys (e.g., [25]). To ensure our target demographic was selected, we applied the participant filter option 'Employment Industry - Software & IT Services'. An additional 13 responses were partially filled (on average at a completion rate of 43.23%). These partially completed responses were included in our analysis since they did yield some insight (see section 7.2). As participants recruited via MTurk have a financial incentive to complete surveys,[8] we ensured strict quality control was applied to each survey response we received. For example, 37 participants opened the survey but did not answer any questions; for this reason, all survey responses by these participants were discarded. We identified that 12 MTurk responses were filled out too quickly (where the median response time was under five minutes; well below the internal average of 20 minutes), and further analysis of these 12 responses indicated poor reading of the question, and thus poor responses; this was identified via our use of alternating positively- and negatively-worded questions. Thus, 12 MTurk responses were removed from the final analysis. Therefore, our final response rate yielded 104 responses of the total 153 participants reached; an overall response rate of 67.97%.

5.1.4 Analysing Response Data

To analyse our response data, we produced a single score for each question's 5-point response. In line with with Brooke's SUS methodology [6], we subtracted one from the raw value of positive items, and subtracted the raw value from five for the negative items. This resulted in values on an ordinal scale of 0–4. We then averaged each response for every question and divide by four (i.e, now a 4-point scale) to obtain scores for each category. For example, two responses of *strongly agree*=5 and one of *neither agree nor disagree*=3 were given to [**A1**] (positively worded); these values are mapped to 4 and 2, respectively, and are averaged (to 3.33) which is then divided by a maximum possible score of four, giving 0.84. We then discretise these calculated values into five intervals (as per table 3, see section 6.1.1) to interpret the findings; this is presented in appendix A under the 'in-practice score' (IPS) for each category.

Demographics for our survey were consistent in terms of the experience levels of developers who responded. 78% of respondents indicated they were professional programmers. Years of programming experience were: <1 year (3.30%); 1–5 years (41.76%); 6–10 years (35.16%); 11–15 years (9.89%); 16–20 years (5.49%); 21–30 years (3.30%); 31–40 years (1.10%); 41+ years (0.00%). A wide range of roles and seniority were listed by developers as presented in fig. 5, thereby indicating that our results include the different expectations of API documentation from a variety of sources. The highest role was a full-stack developer at either a mid-tier or senior role, followed by mid-tier or senior back-end developers and graduate and junior business analysts. Various managerial roles were also listed. Only five students (5.00%) responded in our study, two listing themselves as interns with one as an embedded applications developer. Most respondents were Australian (40.00%), Indian (26.70%) or from the United States (20.00%). Besides information technology services (30.77%), consulting and other software

[5]Our research group's engineering cohort consists of fully-qualified software engineers, with on average 5+ years industry experience.

[6]https://www.addthis.com/ last accessed 7 January 2020.

[7]https://www.mturk.com/ last accessed 9 July 2020.

[8]A total budget of AUD$600 was allocated for recruitment via MTurk, with each participant receiving between AUD$3.50–$10.00.

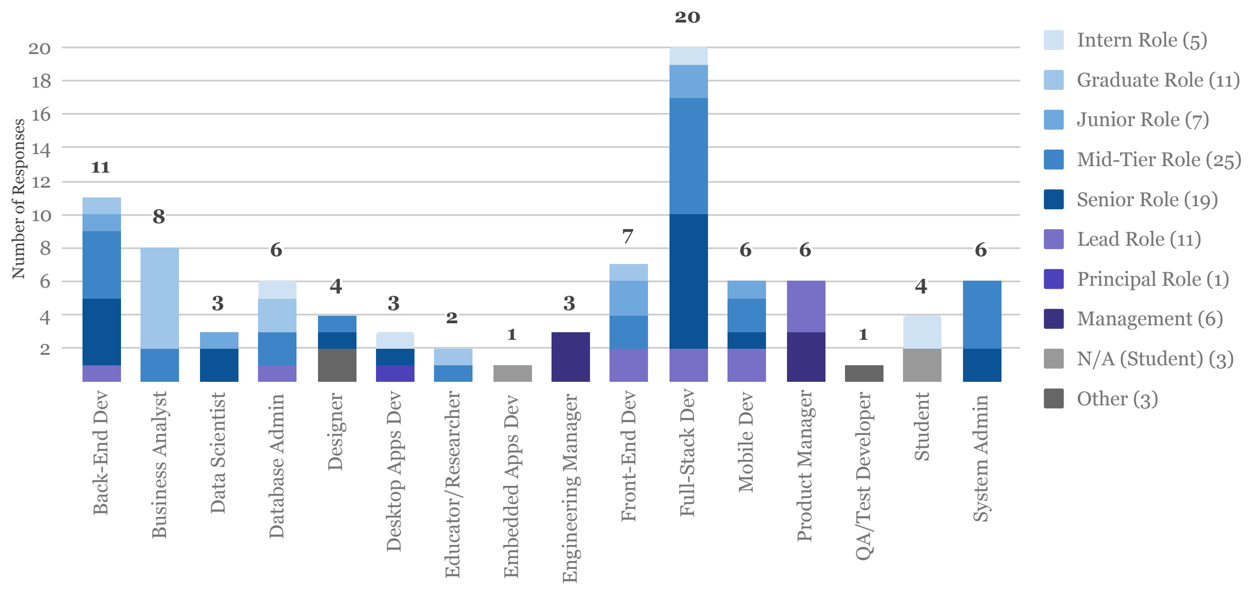


Fig. 5: A wide variety of roles and seniority were observed in our respondents.

development (both at 9.89%) were the most predominant industries listed by participants.

5.2 Empirical Application on Computer Vision Services

Once our taxonomy had been developed and assessed with developers, we performed an empirical application against three computer vision services: Google Cloud Vision [W1], Amazon Rekognition [W2] and Azure Computer Vision [W3]. Our selection criteria in choosing these particular services to analyse is based on the prominence of the service providers in industry and the ubiquity of their cloud platforms (Google Cloud, Amazon Web Services, and Microsoft Azure) in addition to being the top three adopted vendors used for cloud-based enterprise applications [44]. In addition, we had conducted extensive investigation into the services' non-deterministic runtime behaviour and evolution profile in prior work [10] and have also identified developers' complaints about their incomplete documentation in a prior mining study on Stack Overflow [11].

We began with an exploratory analysis of the presence of each dimension and its categories. Appendix B displays all sources of documentation used; although we initially started on the respective services homepages [W1–3], this search was expanded to other webpages hyperlinked. For each category, we listed the documentation's presence as either fully present, partially present or not present at all. This is shown in appendix A with the indication of (half-)filled circles or circle outlines for Google Cloud Vision (abbreviated to GCV), Amazon Rekognition (abbreviated to AWS), and Azure Computer Vision (abbreviated to ACV). Notes were taken for each webpage justifying the presence, and exact sources of documentation were listed when (partially) present. PDFs of each webpage were downloaded between 14–18 March 2019 for analysis. Analysis was performed manually by the lead author by manual inspection of the downloaded web pages (as PDFs) and presence of each item was noted by the lead author using an approach similar to Watson [56].

6 Taxonomy Analysis

In this section, we analyse investigating the taxonomy from two perspectives. Firstly, we contrast the ILS values, being an interpretation of the relevancy researchers have emphasised, against the IPS values found from the results of our survey (being an interpretation of what documentation artefacts developers value more). We are therefore able to identify the API documentation artefacts that are of high value to practitioners, but are yet to be deeply explored by researchers. Secondly, we contrast the IPS values against our assessment of computer vision services, and whether important API documentation artefacts have been included in popular services. We are therefore able to identify whether vendors have or have not already included these highly-valued documentation artefacts within their own APIs, and where existing areas of improvement lie.

6.1 Exploring IPS and ILS Values

6.1.1 IPS Results

IPS values indicate the extent to which developers agree with the statements made in our survey, as calculated by the method described in section 5.1.4. The interpretation of these values are the documentation artefacts (categories) that developers *value* the most. Thus collectively, these artefacts indicate the overall level of importance towards specific API documentation requirements (dimensions).

To interpret these values, we group the data from each of our survey's 34 statements (for each category) into an ordinal scale of five intervals. These intervals indicate relative value to developers; a documentation artefact has *very low* value to developers, *low* value, *medium* value, *high* value, or *very high* value. Table 3 presents these intervals and frequencies of each, with the order of the categories shown in the last column indicating raw IPS values (least useful to most useful) before discretisation in ascending order.

Practitioners tend to agree that each documentation artefact is important to have, and thus IPS values likely fall

TABLE 3: Intervals of ILS (top) and IPS (bottom) values and frequencies.

Research Attention	Range	Frequency	Categories
Very Low	$0.00 \leq \text{ILS}([Xi]) < 0.14$	7	B4, B5, D6, B3, C1, D1, D2
Low	$0.14 \leq \text{ILS}([Xi]) < 0.29$	13	A1, A9, C3, D3, D4, E2, E3, E4, E5, B6, A7, A10, D5
Medium	$0.29 \leq \text{ILS}([Xi]) < 0.43$	9	B2, B7, A4, A12, E1, A3, A8, A11, C2
High	$0.43 \leq \text{ILS}([Xi]) < 0.57$	3	E6, B1, A2
Very High	$0.57 \leq \text{ILS}([Xi]) \leq 0.71$	2	A6, A5
Value to Developers	**Range**	**Frequency**	**Categories**
Very Low	$0.00 \leq \text{IPS}([Xi]) < 0.18$	0	–
Low	$0.18 \leq \text{IPS}([Xi]) < 0.36$	0	–
Medium	$0.36 \leq \text{IPS}([Xi]) < 0.53$	6	D4, B4, C3, C1, E4, B3
High	$0.53 \leq \text{IPS}([Xi]) < 0.71$	16	A4, B6, A2, D2, A6, E2, B5, D6, A8, B2, E6, A10, E5, D5, A9, D3
Very High	$0.71 \leq \text{IPS}([Xi]) \leq 0.89$	12	E3, A7, A3, C2, A12, B1, D1, A11, A1, E1, A5, B7

into the *High* or *Very High* intervals. Only six categories fall into the *Medium* interval and none fall into lower intervals. Developers find technical support contact information [**D4**] to be of the lowest value (see table 3), likely since developers tend to rely on crowd-sourced peer support through mediums such as Stack Overflow. They also see little value in: descriptions of the types of end-users the API is intended for [**B4**]; documentation for non-technical audiences [**C3**]; conceptual information relating the API back to its application domain [**C1**]; structured navigation of the presented API documentation [**E4**]; and descriptions of the intended developers who should be using the API [**B3**].

6.1.2 ILS Results

ILS values indicate overall research attention of categories of our taxonomy through the proportion of papers in our SMS that investigated or reported various issues regarding a specific API documentation artefact. Collectively, each of these categories combined form a dimension (labelled A–E) in a bottom-up approach (see section 3.2.2). Each dimension (top-node) describes the requirements of good quality API documentation, while the category (leaf-node) is the specific API documentation artefact that, collectively, form the requirement. A category with a high ILS value indicates that existing studies that there is substantial attention by researchers on this specific documentation artefact (or, collectively, requirement of good quality API documentation). Conversely, a lower ILS value indicates less attention reported on these categories (artefact) or dimensions (requirement) by the software engineering research community.

To demonstrate the attention of these documentation artefacts within literature, we interpret the ILS values in a similar fashion to the IPS values. It is represented as a discretised value of intervals within a five-dimensional ordinal scale, where the attention on these artefacts in literature are one of: *very low* attention, *low* attention, *medium* attention, *high* attention, *very high* attention. Table 3 indicates the boundaries for each interval (as calculated by the highest ILS value of 0.71 divided by the five intervals) in addition to the frequency of categories appearing in each interval. The order of the categories shown in the last column indicate the ascending order (least research attention to most) of raw ILS values before discretisation. As shown, most of the artefacts (20) found in the taxonomy are discussed in literature disproportionately more than others (i.e., those that fall into the 'low' (13) or 'very low' (7) intervals), though the underlying reasons behind this should be considered on a case-by-case basis (see section 7.3.

There are only five categories that fall into the 'high' or 'very high' intervals, three of which fall under dimension [**A**], Descriptions of API Usage. Research attention on a particular documentation artefact that is considered *Very High* gravitates towards code snippets [**A5**] and tutorials [**A6**]. Code snippets are the readiest form of API documentation for developers, representing exemplary nuggets of information for developers to rapidly digest singular components of the API's functionality. While code snippets generally only reflect small portions of API functionality (generally limited to 15–30 LoC), this is complimented by step-by-step tutorials. These may tie in multiple (disparate) components of API functionality to demonstrate development of more non-trivial applications. Therefore, unsurprisingly, research has substantially explored how best API developers can extract code snippets or write tutorials for these purposes in mind. This is followed by low-level reference documentation [**A2**]—under the 'high' interval—whereby developers should document all client-facing implementation or usage aspects of their API (e.g., class, method, parameter descriptions etc.). Lastly, the entry-level purpose/overview of an API [**B1**] and consistency in the look and feel of the documentation throughout all of the API's official documentation [**E6**] are fall under the 'high' interval. API vendors must give motivation as to why a developer should choose a particular API over another, articulating the *need* of their API, presenting this and other documentation aspects in the easiest way for developers to consume.

6.1.3 Research Opportunities for High-Value Artefacts

In this section, we explore the ILS and IPS values as two distinct indicators of research exploration that would provide the most value to practitioners. We then provide a qualitative discussion by inspecting the intersection of categories at each respective interval identified by our SMS and survey study. Thus, we are able to determine documentation artefacts (categories) and requirements (dimensions) that provide the *greatest value* to developers but have not gained proportional attention in the software engineering literature when compared to other artefacts, and vice-versa. Graphically, we represent these intersections within a five-by-five matrix with intervals of the IPS (x axis) plotted against intervals of the ILS (y axis). Intersections between

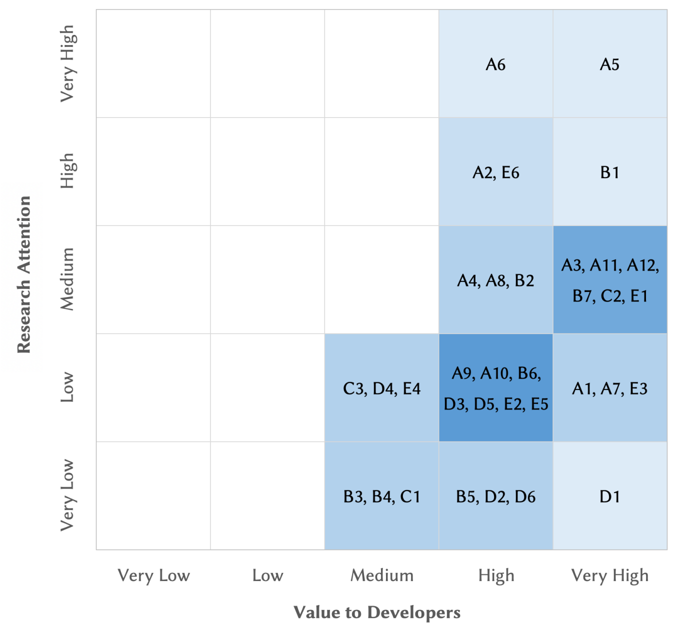


Fig. 6: Value of API documentation artefacts to developers (IPS) vs their research attention (ILS). Colour intensity represents greater number of categories in each intersection

the two are listed for each category within the taxonomy. This is presented in fig. 6.

There is a distinction between (very-)highly valued documentation artefacts whose research attention is (very-)low, as presented in the bottom-right of fig. 6. Most notably, we find that developers find Existence of Support Artefacts [**D**] a highly valued API documentation requirement, but there still exists a substantial gap in existing literature into this requirement. For example, besides category [**D4**] (which is of only *Medium* value to developers), less research has explored all other dimension [**D**] categories (though there may be understandable reasons as to why, as detailed in section 7.3). Furthermore, developers highly value detailed Descriptions of API Usage [**A**] through many documentation artefacts, notably quick-start guides [**A1**], downloadable sample applications [**A7**], exhaustive list of major components [**A9**], and system requirements to use the API [**A10**]. Such artefacts emphasise the need for developers to rapidly pick-up a new API; however, the best ways to provide such information is still open to further investigation in literature.

Conversely, the top-right of fig. 6 emphasises (very)-highly researched artefacts that are of (very)-high value to developers. Here we see that Descriptions of API Usage [**A**] is the most-researched requirement, with code snippets [**A5**] being an API usage artefact that is both most-researched and of highest value. Hence, this demonstrates how many existing studies have an empirical basis on software developers (e.g., via surveys or interviews; see fig. 3)—code snippets is a well-researched artefact since most developers agree to its need in the documentation of APIs. Therefore, it is clear to see how the correlation between the respective ILS and IPS values for [**A5**] are high. However, if we look at other areas of our taxonomy, such as [**A12**], [**B7**], [**D3**], [**E3**] or [**E5**], we find that developers do indeed desire these aspects of API documentation, and, consequently, demand usage descriptions, design rationale descriptions, support artefacts, or good presentation of the documentation to be a necessary requirement of good quality API documentation. Thus, these aspects have not gained proportional attention in literature, thereby highlighting future research potential.

6.2 Triangulating IPS, ILS and Computer Vision

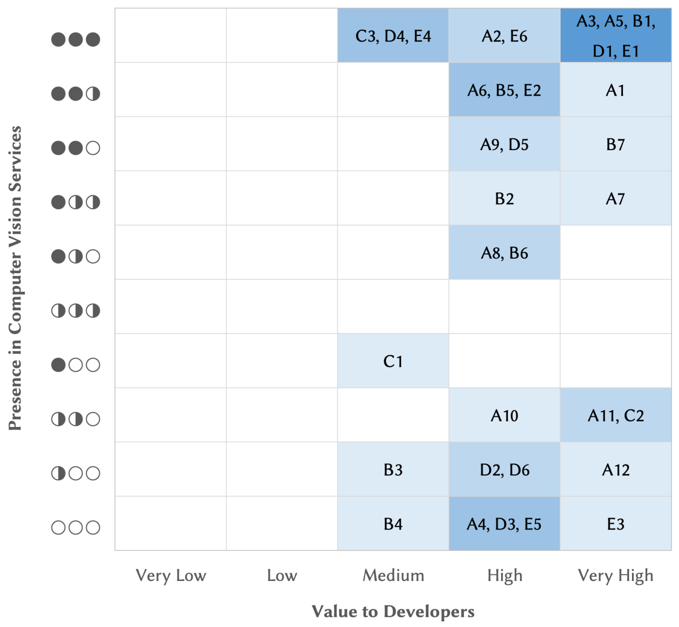


Fig. 7: Value of API documentation artefacts to developers (IPS) vs their presence in computer vision services. Colour intensity represents greater number of categories in each intersection.

To interpret our comparison of IPS values with computer vision services, we introduce a calculated 'presence score' for each category. As discussed in section 5.2, we empirically evaluate each category of our taxonomy with three computer vision services: Azure Computer Vision (ACV), Amazon Rekognition (AWS) and Google Cloud Vision (GCV). We indicate whether the respective API documentation artefact is present, partially present, or nor present (as listed in appendix A). To interpret this data, we assign a full circle (●) for present, half-circle (◐) for partially present and an empty circle (○) for not present. Combinations of presence for each category per service are indicated with the three circles of varying shade. For example, [**A1**] has a presence score of ● ● ◐ because it was found to be present in both GCV and ACV but only partially present in AWS; [**B3**] has a presence score of ◐ ○ ○ because it was only found to be partially present in GCV, etc. For a list of full presence values, see appendix A.

We illustrate which artefacts industry vendors provide developers with and the artefact's respective developer value using this combination of three circles. Using a similar approach to the previous section, these results are presented in a ten-by-five matrix as illustrated in fig. 7. If only one service fully implements a documentation artefact of (very-)high value to developers (● ○ ○), if one or two services partially implement the artefact (◐ ○ ○ and ◐ ◐ ○) or if none do (○ ○ ○), then we believe there is room for improvement for service vendors to improve their documentation and include these artefacts.

In this instance, we can see 10 categories listed in fig. 7 that developers feel are important but are not fully implemented across all three computer vision service vendors. This is especially the case for dimensions [**A**] (Descriptions of API Usage) and [**D**] (Existence of Support Artefacts), corroborating our findings with existing gaps in literature under section 6.1.3. In other words, while both the goals of existing studies and computer vision service vendors have emphasised the need for artefacts such as code-snippets [**A5**], tutorials [**A6**], and entry-points to the API [**B1**], less attention is given to by *both* literature and vendors on the same, (very-)highly valued aspects to developers (e.g., troubleshooting hints [**D2**], licensing information [**D6**] or links to related components [**E3**]).

Furthermore, from our analysis, we can see areas with which the research community has and has *not* paid extensive attention to. We still see that vendors have paid attention to artefacts even where there has been less research attention, namely [**D1**] (FAQs), [**B5**] (success stories), [**A7**] (downloadable sample applications), [**A1**] (quick-start guides), [**E2**] (forums), [**D5**] (printable guides), and [**A9**] (API component lists). These seven categories are of (very) high value to developers but research attention on these topics are (very) low; however, their presence score within computer vision services are ● ◑ ◑ or greater. Hence, we can see that vendors address developer's concerns despite the lack of attention by software engineering researchers in these areas, and thus future research potential to better serve developers and ensure vendors' implementation of these documentation artefacts is evident.

From the above, we can therefore conclude that the vendors' documentation largely covers a majority of API documentation requirements. However, there still remains opportunity for improvement to API documentation by either vendors and/or the research community: that is, low research attention on documentation artefacts that present high value to developers which are *also* generally missing from vendor documentation. To explore this aspect, we triangulate the documentation artefacts (categories) that have a low or very low research attention and that are only present in one service, partially present in one or two, or not present at all. This results in three documentation requirements that warrant further exploration by industry vendors or the research community (see table 4).

6.3 Recommendations Resulting from Analysis

In this section, we triangulate the taxonomy developed from literary sources, the developer survey on this taxonomy to understand its efficacy in-practice, and the application of the taxonomy to computer vision services to provide several recommendations for both service providers and researchers. Our recommendations are based both on extrapolations of our findings, our prior work, and existing experience with such work.

6.3.1 Recommendations for vendors

Table 4 emphasises how service vendors still lack key documentation requirements of critical importance to developers that are still widely under-researched in software engineering literature. The largest of these requirements are the need for vendors to provide additional support artefacts [**D**] and the need for vendors to present this in a way that's most digestible for developers to understand [**E**]. A list of detailed suggestions for vendors are provided in appendix E; here we discuss generalised findings on a sample of key artefacts.

For example, no services assessed had any form of diagrammatic overview of their APIs at a high-level [**D3**], thereby indicating how various components of their APIs work together, such as how specific endpoints work or an overview of the lifecyle of the technical domain behind these endpoints (i.e., label/train/infer/re-train), thereby incorporating conceptual relationships behind the API [**C1**]. For instance, an interactive overview of the developer's need to pre-process their data, send it to the service, and post-process the response data would help developers understand how the service better fits into the 'flow' of their application. Moreover, we failed to find lower-level diagrammatic overviews of the client SDKs—such as a UML diagram—that developers find very useful. We strongly advise vendors to provide diagrams illustrating the service within context to help support existing written documentation.

Troubleshooting hints [**D2**] are also a valuable support artefact, but were only found for AWS's video processing endpoints. As our prior work shows, developers are likely to question what aspects of the service can and cannot do, such as the types of labels it can find, or how to make it focus on specific ontologies when an input image is provided; e.g., time of day (day vs night) location (indoors vs outdoors) or the subject of the image (dog vs cat) [11]. Troubleshooting in identifying service evolution [11] would also be important, since developers are likely to overlook subtle (but application-breaking) changes to response data, such as labels introduced/removed or confidence changes. Therefore, vendors must document detailed troubleshooting suggestions on their websites on how best to resolve discrepancies in the results found from these services. This could easily be tied in with [**A12**] to incorporate usage description requirements when errors are presented to users and how to deal with them; also largely missing from existing documentation.

Another important aspect is the need to make documentation of one component more easily relatable to other parts of the documentation [**E3**]. Again, no service provided quick-links to related documentation; an example here could be links to definitions of domain-specific terminology [**C2**] to help developers with the learning process of adopting these new generation of APIs (e.g., the 'score' field could be linked back to a video explaining the concept of probability within the services' guesses).

6.3.2 Recommendations for researchers

As shown in table 4, we see that there are cases of (very) high-value documentation artefacts (to practitioners) in which literature has not paid great attention to. For example, for the requirement of API usage description [**A**], practitioners agree that both code snippets [**A5**] and documenting system requirements to use the API [**A10**] are of, at least, high value. However, while code snippets has had *consistent*

TABLE 4: Documentation artefacts of high value to developers that have less attention in software engineering literature and are under-documented in computer vision services. Documentation requirements (i.e., dimensions) separated by rules.

Artefact	Value	Research Attention	Presence in Computer Vision Services
[**A10**] Documenting API's minimum system requirements and/or dependencies	High	**Low:** 5 studies (23%)	**Score=1.0:** No dedicated web pages found for this artefact in any service. Dependencies for client libraries embedded within GCV and ACV quick-start guides [W16, 17]. Other system requirements not listed.
[**D2**] Troubleshooting hints	High	**Very Low:** 2 studies (10%)	**Score=0.5:** Only found in AWS's video recognition service [W18], but no troubleshooting tips found for non-video image recognition.
[**D3**] Diagrammatic representation of API	Very High	**Low:** 3 studies (14%)	**Score=0.0:** Not found for any service.
[**D6**] Licensing Information	Very High	**Very Low:** 1 study (5%)	**Score=0.5:** Partially present only in ACV [W19]; information is non-specific to the licensing terms of ACV exclusively.
[**E3**] Quick-links to other relevant components	Very High	**Low:** 3 studies (14%)	**Score=0:** Not found for any service.
[**E5**] Visualised map of navigational paths	Very High	**Low:** 3 studies (14%)	**Score=0:** Not found for any service.

attention within the software engineering research community (i.e., 15 papers spanning 1998–2019), we see that system requirements documentation only gained fluctuating interest by researchers (i.e., predominantly in the 2000s, with two further papers in the last three years). Thus, five papers investigating *some* aspects on this artefact may not cover *all* its aspects; for example, we may have identified a *need* to document these requirements and dependencies, but does this mean we know *all* aspects on how to produce them, the best way to *communicate* them, and the most efficient means for developers to *consume* that information? Contrasting this artefact against the 15 papers on code snippets, we see two documentation artefacts of at least high value to practitioners, yet, evidently, researchers have paid attention to one over the other.

As fig. 6 shows, the need for additional support [**D**] within documentation is the largest requirement that *may* be an indicator for further research in this domain (see section 7.3). Notably, RQ2 of our SMS identified the methodologies and data collection techniques by which our existing understanding of API documentation requirements were gathered; as demonstrated through fig. 3, a majority of our understanding is grounded through the opinions of developers, namely evaluation research using direct techniques. Too many studies are shown to rely on a handful of data collection techniques (interviews and questionnaires, shadowing and observation, think-aloud sessions) and a stronger emphasis for indirect and independent techniques is needed moving forward; there is therefore a gap in literature on *other* types of data collection techniques that may provide different insights into satisfying the documentation requirements within our taxonomy.

For example, we see [**A9**] (exhaustive list of major API components) as a high-value documentation artefact that satisfies the requirement of the API usage description [**A**]. However research attention is lower. A validation research paper could propose a method to generate a baseline list of these components through an independent technique, such mining the API codebase for its major components through class usage (static analysis) or analysing an existing work database or tool use logs to see which components developers have accessed the most. This would satisfy the need for the documentation artefact, bolstering the API usage requirement and exploring new techniques to do so.

Few philosophical papers result in a lack of insight into completely new ways of exploring API documentation. Further exploration into this type of research may help us devise a whole new framework of producing API documentation. For example, as shown by developers and vendors, quick-start guides [**A1**] are highly valued, and well-documented in computer vision services. But literature does not provide any vocabulary or frameworks into how best to develop such guides. Involving both software engineering researchers and developers through a brainstorming or focus group to conceptualise, devise, and refine such a framework may be a worthwhile study to better improve our understanding of quick-start guides whilst also exploring new approaches to research new guidelines.

Beyond requirement [**A**], another insight identified is the need for developers to have visualised maps of navigational paths [**E5**] which is not yet provided by any of the computer vision service providers investigated. With the low ILS value in this category (14% or 3 studies), we see a potential research topic for future exploration. For example, if research can demonstrate that such visualised maps are not just something developers desire, but can make them *more effective* in their day-to-day work, then this could be a strong case made to vendors to improve the presentation of their documentation.

Thus, as we have shown in these sample recommendations, many potential studies and research directions can stem by exploring the discrepancies of API documentation in literature, in practice, and their presence in computer vision services (i.e., as a sample case study) when assessed on a case-by-case basis. The method researchers decide upon depends the research questions they wish to address; thus, observations we present in fig. 3 may trigger fruitful reasoning about approaches future research could take, however inferring methodological gaps will need to be compatible with research goals. Thus, mapping these discrepancies to gaps in the techniques used in studies to devise of novel ways to improve API documentation whilst also exploring new methodologies should be balanced carefully by researchers.

7 Threats to Validity

7.1 Internal Validity

Threats to *internal validity* represent internal factors of our study which affect concluded results. Kitchenham and Charters' guidelines on producing systematic reviews [26] suggest that researchers conducting reviews should discuss the review protocol, inclusion decisions, data extraction with a third party. Within this study, we discussed our protocols with other researchers within our research group and utilised test-retest reliability. Further assessments into reliability would involve an assessment of the review and extraction processes, which can be investigated using inter-rater reliability measures. Guidelines suggested by Garousi and Felderer [14] describe methods for independent analysis and conflict resolution could help resolve this.

As stated in section 3.2, we utilised a systematic SE taxonomy development method by Usman et al. [54]. Two additional taxonomy validation approaches proposed by Usman et al. were not considered in our work: benchmarking and orthogonality demonstration. To our knowledge, there are no other studies that classify existing API documentation studies into a structured taxonomy, and therefore we are unable to benchmark our taxonomy against others. We would encourage the research community to conduct a replication of our work and investigate whether our taxonomy classification approaches are replicable to ensure that categories are reliable and the dimensions fit the objectives of the taxonomy. Moreover, we did not investigate orthogonality demonstration as our primary goals for this work were to investigate the efficacy of the taxonomy by practitioners and in-practice, with reference to our wider research area of intelligent computer vision services. Therefore, we solely adopted the utility demonstration approach in two detailed experiments (sections 5 and 6) to analyse the efficacy of our taxonomy and identify potential improvements for these services' API documentation.

7.2 External Validity

Threats to *external validity* concern the generalisation of our observations. Our systematic mapping study has used a broad range of sources however not all papers contributing to API documentation may have been found or captured within the taxonomy. While we attempted to include as many papers as we could find in our study, some papers may have been filtered out due to our exclusion criteria. For example, there are studies we found that were excluded as they were not written in English, and these excluding factors 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.

Online documentation of APIs are non-static, and may evolve using contributions from both official sources and the developer community (e.g., via GitHub). We downloaded the three service's API documentation in March of 2019—it is highly likely that new documentation may have been added since or modified since publication. A recommendation to mitigate this would be to re-evaluate this study once intelligent computer vision services have matured and become even more mainstream in developer communities.

Unless significant inducements are offered, Singer et al. [51] report that a consistent response rate of 5% has been found in software engineering questionnaires distributed and in information systems the median response rates for surveys are 60% [3]. We observe that low response rates may adversely effect the findings of our survey, typically as software engineers find little time to do them [51]. When compared to typical software engineering studies, our response rate of 67.97% was likely successful due to designing and carefully testing succinct, unambiguous and well-worded questions with researchers within our research group. All adjustments made from the pilot study due to unexpected poor quality of the questionnaire have been reported and explained in section 5.1.2. However, further improvements could be made to increase this response rate.

The survey reached 82 external and 22 internal participants. This yielded a total of 104 participants. However, only 91 participants fully completed the survey and, on average, those who only partially completed the survey completed 43.23% of all questions. Therefore, demographic data for these participants is largely missing. To verify the reliability of partially submitted responses, we calculated the average response of each item in our survey (i.e., question) for all fully completed results and all partially completed results. All partially completed questions, except [**B7**], were within 1 standard deviation from the mean, and therefore we believe the 13 partial results to be valid when excluding B7. Even if these partial results are excluded, our full-response participant count of 91 is still comparable to existing studies, such as Nykaza et al. [40] (57 participants), Robillard and Deline [46] (80 participants), or [45] (83 participants). Therefore, given these comparable numbers, we believe this does not compromise validity of our results.

We also adopt research conducted in the field of questionnaire design, such as ensuring all scales are worded with labels [30] and have used a summating rating scale [52] to address a specific topic of interest if people are to make mistakes in their response or answer in different ways at different times. This approach was also extended using alternating positive and negative sentiment for each question—as multiple studies have shown [48, 7], this approach helps reduce poor-quality responses by minimising extreme responses and acquiescence biases.

7.3 Construct Validity

Threats to *construct validity* relates to the degree by which the data extrapolated in this study sufficiently measures its intended goals. Our interpretation of the ILS (as given in sections 4 and 6.1.2) is reported as the proportion of papers whose research investigates or explores issues regarding the aspects of specific API documentation artefacts (i.e., categories in the taxonomy) that, collectively, comprise the requirements of good API documentation (i.e., dimensions in the taxonomy). Every effort has been made in this work to provide a constructive analysis on the API documentation landscape, however, the studies that comprise the ILS may differ in their intent toward a specific documentation artefact. For example, some studies may have distinct goals to extensively study *how* code snippets [**A5**] specifically improve developer productivity (e.g., through interviews or by observational studies), while others may just reflect that code snippets are a commonly-used artefact self-reported by developers (e.g., through a survey). Thus, the interpretation of the ILS may range between deep exploration of an artefact or whether a study mentions the artefact without any attempts to thoroughly investigate it. For this reason, we suggest that a high ILS value for a category within the taxonomy suggests that the documentation artefact is within the attention of the research community, and that subsequent attention **may** be required for those artefacts with low ILS values as a *potential indicator* for future research (i.e., it also may *not*). However, each artefact with a low ILS (but high IPS) would need to be carefully examined in isolation to evaluate whether future research is indeed warranted, and how that research can be conducted with the ultimate goal to assist practitioners.

Automatic searching was conducted in the SMS by choice of three popular databases (see section 3.1). 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 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.

While we have investigated the application of this taxonomy using a user study (section 5.1), we would like to explore a controlled study of developers to assess how improved and non-improved API documentation impacts developer productivity. The outcome of this work can help design a follow-up experiment, consisting of a comparative controlled study [50] that capture firsthand behaviours and interactions toward how software engineers approach using a computer vision service with and without our taxonomy applied. This can be achieved by providing 'mock' improved documentation with the suggested improvements included in this work. Such an experiment could recruit a sample of developers of varying experience (from beginner programmer to principal engineer) to complete a certain number of tasks under a comparative controlled study, half of which will (a) develop using the improved 'mock' documentation, and the other half will (b) develop with the *as-is/existing* documentation. From this, we can compare if the taxonomy makes improvements by capturing metrics and recording the sessions for qualitative analysis. Visual modelling can be adopted to analyse the qualitative data using matrices [12], maps and networks [49] as these help illustrate any causal, temporal or contextual relationships that may exist to map out the developer's mindset and difference in approaching the two sets of designs of the same tasks.

8 Conclusions & Future Work

The emergence of AI-based intelligent components present significant challenges to our existing understanding of traditional API documentation. The inherent probabilistic and non-deterministic nature of these components means that developers must shift their mindset of conventional APIs, and vendors of these services must similarly shift the mindset of documenting their APIs using traditional means. Without adapting to the new mental model (of the vendors designing these services) and by vendors presenting poor or incomplete (traditional) documentation that is not compatible with these next-generation components, developers face many struggles. They fail to grasp how to properly understand how these services work, seeking further documentation or support from their peers on forums on such as Stack Overflow [11]. This ultimately hinders developers' productivity and thus adversely affects the internal quality of the applications that they build.

This study has explored the artefacts and means by which traditional API documentation is studied through the use of an SMS of 4,501 studies, identifying 21 key works. From this, we synthesised a taxonomy of the various documentation artefacts that improves API documentation quality, and thus collectively synthesising the requirements of good API documentation. Furthermore, we also capture the most commonly used analysis techniques used in the academic literature to understand the means by which the goals of these studies resulted in their findings. We then validate our taxonomy against developers to assess its efficacy with practitioners, and conduct a heuristic evaluation against three popular computer vision services. We determine that developers demand certain documentation artefacts more than others, since not all documentation artefacts are equally valued. We map the value (to developers) of these artefacts against their exposure within the software engineering literature, thereby highlighting the gaps by which future research could expand upon. Furthermore, we present a similar mapping against how well the coverage computer vision services have incorporated such artefacts into their own API documentation, thus highlighting that while industry vendors cover most documentation artefacts that may not be in the interest to researchers, some artefacts with low research interest are still largely missing (see table 4). We therefore provide several generalised recommendations to vendors and the wider research community to explore how best these artefacts can be better addressed and incorporated into further research, thus improving our understanding of the requirements of good API documentation.

Future extensions of our work may involve a restricted systematic literature review in API documentation artefacts, and many suggestions are further detailed in section 7. Further, a review into the techniques of these primary studies may extend the mapping we conducted in this work, by evaluating the the effectiveness of the various approaches used in each study and assessing these against the proposed conclusions of each study.

The findings of our work provides a solid baseline for improving the documentation of non-deterministic software, such as computer vision services. While our aim is to eventually improve the quality of API documentation, the ultimate goal is to improve the software engineer's experience of non-deterministic and abstracted AI-based components, such as intelligent web services. We hope the guidelines from this extensive study help both software developers and API providers alike by using our taxonomy as a go-to checklist for what should be considered in documenting any API.

Acknowledgements

Cummaudo is supported by an Australian Government Research Training Program (RTP) Scholarship. Grundy is supported by Laureate Fellowship FL190100035. We acknowledge additional support provided by the ARC Research Hub for Digital Enhanced Living IH170100013.

References

[1] E. Aghajani, C. Nagy, O. L. Vega-Marquez, M. Linares-Vasquez, L. Moreno, G. Bavota, and M. Lanza, "Software Documentation Issues Unveiled," in *Proceedings of the 41st International Conference on Software Engineering*. Montreal, QC, Canada: IEEE, May 2019, pp. 1199–1210.

[2] L. Aversano, D. Guardabascio, and M. Tortorella, "Analysis of the Documentation of ERP Software Projects," *Procedia Computer Science*, vol. 121, pp. 423–430, January 2017.

[3] Y. Baruch, "Response rate in academic studies - A comparative analysis," *Human Relations*, vol. 52, no. 4, pp. 421–438, 1999.

[4] C. Bottomley, "What part writer? What part programmer? A survey of practices and knowledge used in programmer writing," in *Proceedings of the 2005 IEEE International Professional Communication Conference*. Limerick, Ireland: IEEE, July 2005, pp. 802–812.

[5] 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, April 2007.

[6] J. Brooke, "SUS-A quick and dirty usability scale," *Usability Evaluation in Industry*, pp. 189–194, 1996.

[7] ——, "SUS: a retrospective," *Journal of Usability Studies*, vol. 8, no. 2, pp. 29–40, 2013.

[8] D. V. Cicchetti, "Guidelines, Criteria, and Rules of Thumb for Evaluating Normed and Standardized Assessment Instruments in Psychology," *Psychological Assessment*, vol. 6, no. 4, pp. 284–290, 1994.

[9] A. Cummaudo, R. Vasa, and J. Grundy, "What should I document? A preliminary systematic mapping study into API documentation knowledge," in *Proceedings of the 13th International Symposium on Empirical Software Engineering and Measurement*. Porto de Galinhas, Recife, Brazil: IEEE, October 2019, pp. 1–6.

[10] A. Cummaudo, R. Vasa, J. Grundy, M. Abdelrazek, and A. Cain, "Losing Confidence in Quality: Unspoken Evolution of Computer Vision Services," in *Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution*. Cleveland, OH, USA: IEEE, December 2019, pp. 333–342.

[11] A. Cummaudo, R. Vasa, S. Barnett, J. Grundy, and M. Abdelrazek, "Interpreting Cloud Computer Vision Pain-Points: A Mining Study of Stack Overflow," in *Proceedings of the 42nd International Conference on Software Engineering*. Seoul, Republic of Korea: IEEE, May 2020, In Press.

[12] I. Dey, *Qualitative Data Analysis: A User-Friendly Guide for Social Scientists*. New York, NY: Routledge, 1993.

[13] M. Gamer, J. Lemon, I. Fellows, and P. Singh, "Irr: various coefficients of interrater reliability," *R package version 0.83*, 2010.

[14] 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*, vol. Part F1286. Karlskrona, Sweden: ACM, June 2017, pp. 170–179.

[15] 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.

[16] R. S. Geiger, N. Varoquaux, C. Mazel-Cabasse, and C. Holdgraf, "The Types, Roles, and Practices of Documentation in Data Analytics Open Source Software Libraries: A Collaborative Ethnography of Documentation Work," *Computer Supported Cooperative Work: CSCW: An International Journal*, vol. 27, no. 3-6, pp. 767–802, May 2018.

[17] R. L. Glass, I. Vessey, and V. Ramesh, "RESRES: The story behind the paper "Research in software engineering: An analysis of the literature"," *Information and Software Technology*, vol. 51, no. 1, pp. 68–70, 2009.

[18] K. A. Hallgren, "Computing Inter-Rater Reliability for Observational Data: An Overview and Tutorial," *Tutorials in Quantitative Methods for Psychology*, vol. 8, no. 1, pp. 23–34, February 2012.

[19] S. Haselbock, R. Weinreich, G. Buchgeher, and T. Kriechbaum, "Microservice Design Space Analysis and Decision Documentation: A Case Study on API Management," in *Proceedings of the 11th International Conference on Service-Oriented Computing and Applications, SOCA 2018*, Paris, France, November 2019, pp. 1–8.

[20] 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*, ser. questions and tradeoffs with API documentation for C++ projects. Gothenburg, Sweden: ACM, May 2018, pp. 643–653.

[21] H. Hosseini, B. Xiao, and R. Poovendran, "Google's cloud vision API is not robust to noise," in *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications*, vol. 2017-Decem. Cancun, Mexico: IEEE, December 2017, pp. 101–105.

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

[23] S. Inzunza, R. Juárez-Ramírez, and S. Jiménez, "API Documentation," in *Proceedings of the 6th World Conference on Information Systems and Technologies*. Naples, Italy: Springer, March 2018, pp. 229–239.

[24] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, "Improving documentation for eSOA APIs through user studies," in *Proceedings of the First International Symposium on End User Development*, vol. 5435 LNCS. Siegen, Germany: Springer, March 2009, pp. 86–105.

[25] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 5589, no. c, pp. 1–1, 2020.

[26] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Software Engineering Group, Keele University and Department of Computer Science, University of Durham, Keele, UK, Tech. Rep., 2007.

[27] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 3, pp. 63–92.

[28] A. J. Ko and Y. Riche, "The role of conceptual knowledge in API usability," in *Proceedings of the 2011 IEEE Symposium on Visual Languages and Human Centric Computing*. Pittsburg, PA, USA: IEEE, September 2011, pp. 173–176.

[29] J. Kotula, "Using patterns to create component documentation," *IEEE Software*, vol. 15, no. 2, pp. 84–92, 1998.

[30] J. A. Krosnick, "Survey Research," *Annual Review of Psychology*, vol. 50, no. 1, pp. 537–567, February 1999.

[31] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, p. 159, March 1977.

[32] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical Software Engineering*, vol. 10, no. 3, pp. 311–341, July 2005.

[33] W. Maalej and M. P. Robillard, "Patterns of knowledge in API reference documentation," *IEEE Transactions on Software Engineering*, 2013.

[34] S. G. McLellan, A. W. Roesler, J. T. Tempest, and C. I. Spinuzzi, "Building more usable APIs," *IEEE Software*, vol. 15, no. 3, pp. 78–86, 1998.

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

[36] M. Meng, S. Steinhardt, and A. Schubert, "Application programming interface documentation: What do software developers want?" *Journal of Technical Writing and Communication*, vol. 48, no. 3, pp. 295–330, August 2018.

[37] B. A. Myers, A. J. Ko, T. D. LaToza, and Y. Yoon, "Programmers are users too: Human-centered methods for improving programming tools," *Computer*, vol. 49, no. 7, pp. 44–52, 2016.

[38] B. A. Myers and J. Stylos, "Improving API Usability," *Communications of the ACM*, vol. 59, no. 6, p. 62–69, May 2016.

[39] K. Nybom, A. Ashraf, and I. Porres, "A systematic mapping study on API documentation generation approaches," in *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*. Prague, Czech Republic: IEEE, August 2018, pp. 462–469.

[40] J. Nykaza, R. Messinger, F. Boehme, C. L. Norman, M. Mace, and M. Gordon, "What programmers really want: Results of a needs assessment for SDK documentation," in *Proceedings of the*

20th Annual International Conference on Computer Documentation. Toronto, ON, Canada: ACM, October 2002, pp. 133–141.

[41] D. L. Parnas and S. A. Vilkomir, "Precise documentation of critical software," in *Proceedings of 10th IEEE International Symposium on High Assurance Systems Engineering*. Plano, TX, USA: IEEE, November 2007, pp. 237–244.

[42] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008*, 2008, pp. 68–77.

[43] R Core Team, *R - A Language and Environment for Statistical Computing*, https://www.R-project.org/, R Foundation for Statistical Computing, Vienna, Austria, 2020.

[44] RightScale Inc., "State of the Cloud Report: DevOps Trends," Tech. Rep., 2016.

[45] M. P. Robillard, "What makes APIs hard to learn? Answers from developers," *IEEE Software*, vol. 26, no. 6, pp. 27–34, 2009.

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

[47] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosall, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong, "On-demand developer documentation," in *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution*. Shanghai, China: IEEE, September 2017, pp. 479–483.

[48] J. Sauro and J. R. Lewis, "When designing usability questionnaires, does it hurt to be positive?" in *Proceedings of the 2011 SIGCHI Conference on Human Factors in Computing Systems*, Vancouver, BC, Canada, May 2011, pp. 2215–2223.

[49] T. A. Schwandt, "Qualitative data analysis: An expanded sourcebook," *Evaluation and Program Planning*, vol. 19, no. 1, pp. 106–107, 1996.

[50] C. B. Seaman, "Qualitative methods," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 2, pp. 35–62.

[51] J. Singer, S. E. Sim, and T. C. Lethbridge, "Software engineering data collection for field studies," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 1, pp. 9–34.

[52] P. Spector, *Summated Rating Scale Construction*. Newbury Park, CA, USA: SAGE, 1992.

[53] A. Taulavuori, E. Niemelä, and P. Kallio, "Component documentation - A key issue in software product lines," *Information and Software Technology*, vol. 46, no. 8, pp. 535–546, June 2004.

[54] 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.

[55] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With great training comes great vulnerability: Practical attacks against transfer learning," in *Proceedings of the 27th USENIX Security Symposium*. Baltimore, MD, USA: USENIX Association, July 2018, pp. 1281–1297.

[56] R. 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*. Seattle, WA, USA: ACM, October 2012, pp. 295–302.

[57] R. Watson, M. Mark 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*. Greenville, SC, USA: ACM, September 2013, pp. 165–174.

[58] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requirements Engineering*, vol. 11, no. 1, pp. 102–107, March 2006.

[59] 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.

Alex Cummaudo is an R&D Software Engineer and holds an Associate Research Fellowship at the Applied Artificial Intelligence Institute (A^{2}I^2), Deakin University, Australia. Alex received a BIT(Hons) in 2017 from Deakin, where he developed a novel computer vision pipeline to detect alphanumeric sequences in unstructured images, and a BSc in 2016 from Swinburne University of Technology, Australia. He is interested in abstracting pre-trained machine learning models through software components, and explores ways to enhance developer productivity and DevX of AI-based software. Alex is working towards completing his PhD at A^{2}I^2, where he developed a novel software architecture tactic to improve integration reliability of AI-based components with traditional software, based on a thorough investigation of the behavioural and evolutionary profile of intelligent web services.

Rajesh Vasa is a professor of software and technology innovation at Deakin University and currently leads transnational research at the Applied Artificial Intelligence Institute. He has more than 2 decades of experience spanning both industry and academia with deep skills in data science, artificial intelligence and complex software systems design. His career spans roles in development, operations and executive leadership in projects and organisations across the world. Recent work included building intelligent homes for aged care, reducing traffic congestion, deep learning and neural networks in healthcare, and automating the process of innovation. In the context of software engineering, his focus is on robust AI systems, sustainable software evolution, and software architecture.

John Grundy is Australian Laureate Fellow and Professor of Software Engineering at Monash University, Australia. He has published widely in automated software engineering, domain-specific visual languages, model-driven engineering, software architecture, and empirical software engineering, among many other areas. He is Fellow of Automated Software Engineering and Fellow of Engineers Australia.

Mohamed Abdelrazek is Associate Professor of Software Engineering and IoT. Mohamed has more than 15 years of the software industry, research and teaching experience. Before joining Deakin University in 2015, Mohamed worked as a senior research fellow at Swinburne University of Technology and Swinburne-NICTA software innovation lab (SSIL). Mohamed moved to Australia in 2010 to do his PhD in cloud computing security. Before that, Mohamed was the head of software development department at Microtech, a large software house, managing large-scale products/projects and managing large teams. Mohamed has deep experience in designing, developing, integrating, and managing large-scale software systems. Mohamed's research interests include Automated Software Engineering for Artificial Intelligence, Software Security, and Human-Centric Design.