

API Documentation-Related Problems: A Systematic Mapping Study

Gabriel Guimarães de Almeida

dealmeida.gabriel.d@gmail.com
gabriel.almeida8@estudante.ufla.br
Universidade Federal de Lavras
Lavras, Minas Gerais

Mauricio Ronny de Almeida Souza

mauricio.ronny@ufla.br
Universidade Federal de Lavras
Lavras, Minas Gerais

Fischer Jônatas Ferreira
fischer.ferreira@unifei.edu.br

Universidade Federal de Itajubá
Itajubá, Minas Gerais

ABSTRACT

The growing reliance on Application Programming Interfaces (APIs) in modern systems architecture makes the quality of their documentation critical for abstracting complexity and ensuring effective consumption by developers. However, API documentation frequently suffers from issues arising from maintenance and prioritization failures, leading to operational difficulties, integration errors, and systemic malfunctions. This work presents a Systematic Mapping Study (SMS) focused on identifying and categorizing problems, impacts, and mitigation strategies related to API documentation. The primary study search stage retrieved 314 studies from the Scopus database, which, after filtering and snowballing, resulted in the selection of 61 studies. The mapping revealed 26 distinct problems organized into four categories: “Content”, “Presentation”, “Usability”, and “Sustainability”. From the identified problems, 20 correlated impacts were characterized, exposing a cascading causal relationship where certain impacts manifest as new problems. In light of the magnitude of these challenges, the literature demonstrates a limitation in mitigation strategies, covering only 8 types of actions, which often lack specificity. The results highlight a gap in practical solutions and provide a consolidated foundation to guide future research and the development of more effective tools for the API ecosystem.

KEYWORDS

API, Systematic Mapping Study, Documentation, Problems, Impacts, Actions

1 INTRODUCTION

In software development, APIs (“Application Programming Interface”) are fundamental for providing functionalities and resources to consumers [17, 22, 25, 52, 57], abstracting complex workflows and promoting code reuse [17, 22, 31, 34, 57, 63]. Whether in local libraries [34] or on the Web [23, 63], the correct utilization of an API requires the comprehension of its documentation [56, 65], an essential artifact for extracting

operational information [9, 25, 52, 56]. However, the quality of the documentation varies significantly; high quality attracts users, whereas deficient documentation discourages use or leads to the abandonment of the technology [55]. In the context of modern Web development, where applications heavily depend on the integration of multiple third-party REST APIs and microservices, documentation quality becomes a critical factor for successful service composition and consumption [12, 23, 63].

This paper presents a Systematic Mapping Study (SMS) [44] investigating the state of the art on problems related to API documentation reported between 2016 and 2025. We purposely adopt the term “problems related to API documentation” to address not only the issues within the documentation itself but also the challenges surrounding its usage context. This broader scope encompasses the absence of documentation, difficulties in maintenance or access, and other systemic issues that extend beyond its textual content.

The methodology combines a targeted search string with snowballing to ensure a wide literature coverage, resulting in a final selection of 61 primary studies. Following a rigorous selection process based on well-defined inclusion and exclusion criteria, selected papers underwent full-text analysis. Our findings reveal 26 distinct problems organized into four categories (“Content”, “Presentation”, “Usability”, and “Sustainability”) and characterize 20 correlated impacts, exposing a cascading causal relationship where certain impacts manifest as new problems. Furthermore, the mapping highlights a critical gap in current literature: mitigation strategies cover only 8 types of actions, which frequently lack specificity. Ultimately, these results provide a consolidated foundation to bridge this shortage of practical solutions and guide future research and tool development within the API ecosystem.

As a contribution, this work provides: (1) an updated and extended taxonomy that expands the original set of categories proposed by Uddin and Robillard [55] with two new dimensions, capturing operational and contextual obstacles; (2) the academic community and software professionals with an updated knowledge base on problems related to API documentation for the conscious construction and consumption of documentation. The consolidation of problems, impacts, and mitigations aims to reduce comprehension bottlenecks and mitigate the incorrect use of APIs, increasing efficiency in Software Engineering.

In: Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia’2026). Lavras/MG, Brazil. Porto Alegre: Brazilian Computer Society, 2026.

© 2026 SBC – Brazilian Computing Society.
ISSN 2966-2753

The remainder of this article is structured as follows: Section 2 presents the theoretical background and related work; Section 3 details the SMS methodology; Section 4 exposes the obtained results; Section 5 discusses the findings; Section 6 points out the limitations; and Section 7 closes the paper with conclusions and future work.

2 THEORETICAL BACKGROUND

This Section presents the fundamental concepts for understanding this work and the comparative mapping of the main related studies.

2.1 APIs and Their Documentation

In Software Engineering, the use of APIs is a consolidated practice to abstract the complexity of specific implementations and promote code reuse [18, 32, 37, 38, 54]. However, their effective utilization requires understanding their architectural diversity, since the interaction format is intrinsically linked to their type [34, 56, 66].

A common format is Library APIs [5, 34], which are locally integrated into the project via dependency managers. Listing 1 illustrates this model using the *NumPy* library¹ in *Python*. In contrast, Web APIs are made available over a network, requiring a remote connection and specific communication protocols [3, 43], as exemplified by consumption via a cURL request in Listing 2 and via an interactive *Swagger* documentation².

Listing 1: Example of usage of NumPy API

```
import numpy as np
matrix = np.array([[1, 2, 3], [3, 2, 1]])
```

Listing 2: Example of usage of a Web API

```
curl --location 'https://randomuser.me/api/'
```

Regardless of the architecture, the correct and effective use of an API requires a complete understanding of its documentation, which acts as a formal contract between provider and consumer. High-quality documentation must detail input parameters, return structures, business rules, and practical usage examples. This work aims to map problems in documentations of any API nature.

2.2 Definition of Problems, Impacts, and Actions Relate to API Documentation

To guide data extraction and the analysis of the results of this SMS, three central concepts were defined: **Problems**: Deficiencies, failures, gaps, or anti-patterns described directly or indirectly in the content, presentation, maintenance, or context of a documentation (e.g., the lack of format standardization pointed out by Cao et al. [10]). **Impacts**: Negative consequences resulting from documentation-related problems, affecting usability, users, and the quality of the final artifact (e.g., the finding by Nybom et al. [41] that developers who

resort to *StackOverflow* instead of official documentation produce less secure code). **Proposed Actions**: Solutions or mitigations presented in the literature to bypass such problems (e.g., the combination of manual writing and automated contract extraction suggested by Pandita et al. [42]).

2.3 Related Work

In the scope of API documentation problems, the works of Uddin and Robillard [55] and Ajam et al. [2] stand out. The former, published in 2015, compiled 10 failures categorized into “Content” and “Presentation”. Uddin and Robillard [55] explores problems from the perspective of documentation analysis and practitioners’ opinions. However, the authors do not compare their results with the existing literature. The latter, from 2021, explores key quality dimensions of API documentation in existing literature. However, Ajam et al. [2] do not explore the specific problems related to those quality dimensions. Both papers also lack the discussion of the impacts caused by API documentation problems, and actions that are suggested to overcome or to mitigate the problems. Finally, the problems and quality dimensions observed by Ajam et al. [2], Uddin and Robillard [55] are confined to the content and presentation of the API documentation itself, and do not consider problems that affect the context of its usage, e.g: the lack of documentation.

Therefore, the current paper aims to expand the catalog proposed by Uddin and Robillard [55], building upon their initial taxonomy, but extending it. The following categories are adopted to classify the findings of this study: “**Content**” and “**Presentation**”: Focus, respectively, on the textual/technical information provided and on the visual/structural way they are exposed. “**Usability**” and “**Sustainability**”: Act, respectively, on the operational constraints caused to the developer and on the health, evolution, and long-term maintenance of the artifact.

3 METHOD

According to Petersen et al. [44], Systematic Mapping Study is a secondary study method designed to provide a comprehensive overview of a research area, structuring the field by identifying, categorizing, and quantifying existing literature to uncover trends and research gaps. In this paper a SMS was executed in three main stages, detailed below: research questions formulation (Subsection 3.1), search strategy and study selection (Subsection 3.2), and mapping of the extracted information (Subsection 3.3).

3.1 Research Questions Formulation

To guide data extraction and identify gaps, challenges, and solutions regarding the topic, three research questions were formulated:

- **RQ1.** *What are the problems related to API documentation?* – Aims to map recurring gaps and difficulties faced in the literature by practitioners in the field.
- **RQ2.** *What are the main impacts, or consequences, related to the problems raised in RQ1?* – Analyzes

¹<https://numpy.org/devdocs/reference/index.html>

²<https://swagger.io/>

how such failures negatively influence productivity and quality in software development.

- **RQ3.** *What are the actions proposed in the literature to solve these documentation problems?* – Identifies solutions and recommended approaches to mitigate the mapped problems.

3.2 Search and Selection of Studies

To search for primary studies, we used the search *string* presented in Listing 3, submitted to the *Scopus* database³, due to its broad coverage in Software Engineering, on the first semester of 2025. We selected *Scopus* as the single search source because it indexes the proceedings of major software engineering venues (including ACM and IEEE conferences). While this introduces a single-source limitation (discussed in Section 6), we applied systematic snowballing to expand coverage beyond database boundaries. Then, the period from 2016 to 2025 was filtered, initially resulting in 314 articles.

Listing 3: String search

```
(( api ) AND ( problem OR challenge OR
    difficult OR issue ) AND documentation)
```

For study selection, the following Exclusion Criteria (EC) were applied:

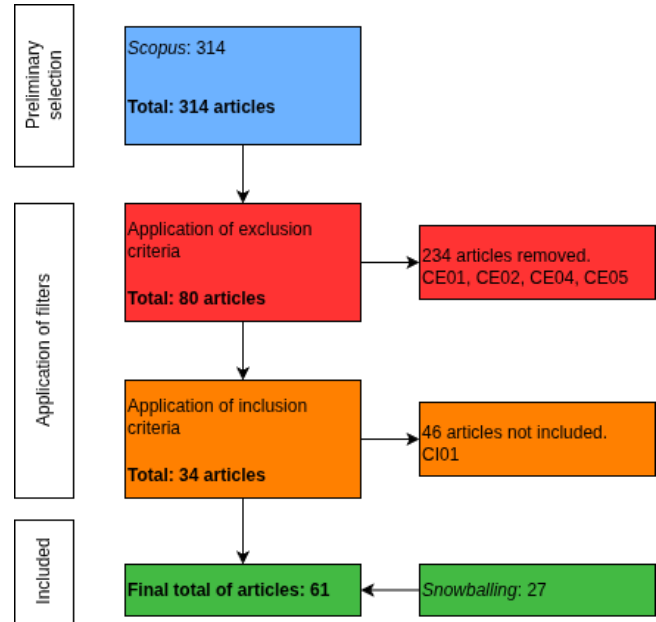
- **EC01.** The paper does not address API documentation within the context of software engineering;
- **EC02.** Published in a language other than Portuguese or English;
- **EC03.** Publication type differs from conference or journal article;
- **EC04.** Documentary inaccessibility (unavailable for download via the CAPES portal or under a *paywall*);
- **EC05.** Duplicate paper;
- **EC06.** Work in progress (not completed).

Following the guidelines of Petersen et al. [44], quality assessment was not applied as a filtering criterion in this systematic mapping study. Unlike systematic literature reviews (SLRs), which aim to synthesize evidence and therefore require quality appraisal, SMS studies focus on classifying and structuring the research landscape. Nevertheless, we acknowledge that incorporating quality assessment could strengthen the validity of frequency-based claims, and this remains an opportunity for future work.

The ECs were applied through the reading of titles and abstracts, disregarding 234 studies and leaving 80. To this amount, the single Inclusion Criterion was applied (**IC01:** *The paper explores API documentation problems and challenges in software engineering*), evaluated by reading the introduction of the works, which resulted in 34 articles. Finally, a *snowballing* process was performed, applying all exclusion and inclusion criteria to the references of each one of the included works, then adding 27 new studies. The complete selection workflow, which totaled 61 accepted articles, is summarized in Figure 1.

³<https://www.scopus.com/home.uri?zone=header&origin=sbrowse>

Figure 1: Representation of the article selection stages



3.3 Data Extraction and Mapping

Data extraction was conducted through a full analytical reading of the 61 selected primary studies to identify and record problems, impacts, and mitigating actions related to API documentation.

To ensure rigor and minimize bias, the screening, full-text assessment, and data extraction were independently performed by three researchers. The team met weekly to review findings and reach a consensus on the data extracted from each paper, resolving any disagreements through collaborative discussion. To answer the research questions, we synthesized the qualitative data using Thematic Analysis [13] combined with the Card Sorting technique [50].

This method allowed us to systematically group the extracted problems (RQ1), extending the initial taxonomy proposed by Uddin and Robillard [55], as well as map all reported impacts (RQ2) and actions (RQ3) while explicitly establishing their causal relationships. Finally, the resulting card sorting categories were validated through full consensus among all three researchers to ensure categorization consistency.

4 RESULTS

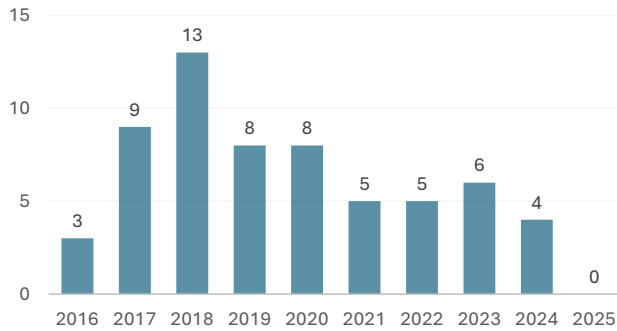
This section presents the analysis of the findings and the answers to the research questions (RQ), structured into four subsections: a general data analysis (Section 4.1) and the specific answers for RQ1 (Section 4.2), RQ2 (Section 4.3), and RQ3 (Section 4.4).

4.1 General Data Analysis

The temporal analysis (Figure 2) reveals a peak of interest in the topic in 2018 (13 publications), followed by a progressive

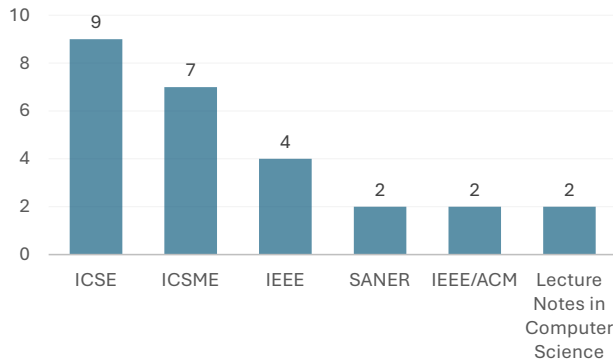
decline that culminated in four articles in 2024. Although no articles were retrieved for 2025 (up to the date of the search execution), the historical trend from 2018 to 2024 suggests a reduction in the literature's focus. Possible causes are the consolidation of *Swagger* and the advancement of Artificial Intelligence for generating examples, or, alternatively, reflects a low awareness within research regarding the persistence of these problems in developers' daily routines.

Figure 2: Publications per year



Regarding publication venues (Figure 3), **ICSE** (9 articles) and **ICSME** (7 articles) stand out, followed by the *IEEE TSE* journal (4 articles), and by *SANER*, *IEEE/ACM ASE*, and *LNCS*, with two publications each. These data consolidate the channels of greatest relevance and historical background for submissions in the area.

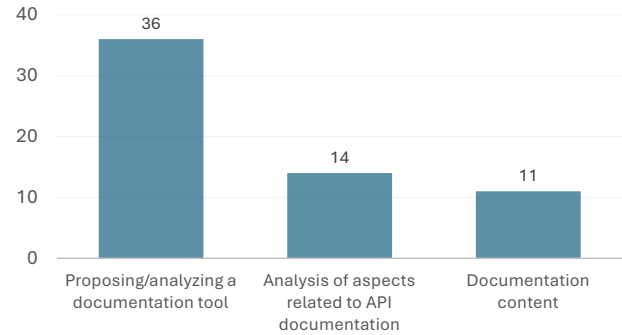
Figure 3: Publications per venue



All selected articles mentioned documentation problems, even if this was not their central focus. Figure 4 details the thematic focus of the studies: 36 were dedicated to proposing or analyzing tools [2, 4, 6, 7, 9–11, 16, 19, 22–25, 29, 31–33, 38, 39, 42, 43, 45, 46, 51, 53, 54, 56–62, 64–66]; 14 focused on the analysis of general aspects of documentation [3, 12, 15, 19, 20, 30, 35–37, 40, 41, 49, 52, 63] and 11 evaluated content strictly [1, 5, 8, 14, 17, 18, 21, 26, 27, 34, 47], evidencing a scientific gap in this latter research line. It is important to

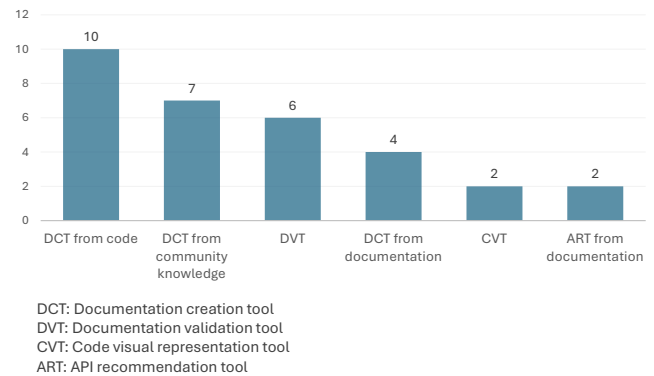
highlight that these categories are not mutually exclusive. Studies that addressed more than one theme were classified according to their primary contribution.

Figure 4: Publications per approach



Among the researches focused on tools (Figure 5), the most frequent type automates documentation creation from code (10 articles) [11, 17, 23, 24, 29, 34, 39, 45, 56, 66], followed by generation via community knowledge (7 articles) [2, 19, 46, 51, 54, 57, 61]. Quality validation tools accounted for 6 articles [4, 22, 25, 33, 64, 65], while creation based on another documentation obtained 4 [7, 10, 42, 59]. Visual representation and recommendation systems recorded only two articles each [32, 60, 62, 66], pointing out potential research opportunities.

Figure 5: Publications per type of tool



4.2 RQ1. What are the problems related to API documentation?

We adopted and expanded the taxonomy of problems proposed by Uddin and Robillard [55], and identified 26 problems, organized in four categories:

- **Content.** Problems directly related to the API documentation content;

- **Presentation.** Problems related to how the API documentation content is presented to the reader.
- **Usability.** Problems related with how the reader, or developer, uses the API documentation;
- **Sustainability.** Problems related to how the API documentation is maintained.

The categories “Content” (accuracy and sufficiency) and “Presentation” (visual layout) are defined by Uddin and Robillard [55], however we identify a set of problems, such as inexistence, difficulty of access or duplication, that demanded the creation of the two new categories: “Usability” (obstacles in artifact consumption) and “Sustainability” (threats to longevity and maintenance). Table 1 maps the problems to the primary studies from which they were observed. The problems are organized by categories and ordered by occurrence. Shaded rows in Table 1 indicate new problems in relation to Uddin and Robillard [55].

The top five recurring problems are “Incompleteness” (“Content”), “Compromised maintenance” (“Sustainability”), “Incorrectness” (“Content”), “Nonexistence” (“Usability”) and “Human-only” (“Content”). The problems “Incompleteness”, “Incorrectness”, and “Compromised maintenance” are presented as quality dimensions on their own by Ajam et al. [2], respectively: “Completeness”, “Correctness”, and “Maintainability”. “Incompleteness” stands out as the most recurring problems both in our results and in the results of Uddin and Robillard [55]. The other quality dimensions mapped by Ajam et al. [2] are “Consistency” and “Understandability”, which can be mapped to the problems “Inconsistency” and “Hard comprehension”, of our results. “Hard comprehension” is a problem not originally mapped by Uddin and Robillard [55].

“Content” is the dominant category in our results. We confirmed all the original content problems identified by Uddin and Robillard [55], and found four new problems: “Human-only”, “Hard comprehension”, “Bias”, and “Brevity”. Conceptually, the “Bias” and “Human-only” problems share a fundamental similarity, as both stem from a restricted design perspective regarding their target audience. While “Bias” limits the documentation to a single, specific developer viewpoint (often neglecting diverse user needs) “Human-only” restricts the artifact entirely to human reading, lacking the structured formats required for consumption by automated tools. Despite the problem “Hard comprehension” impacting the usability of the API, we categorized this problem in the category “Content”, because the solution to its problem is related to adjusting the content of the documentation. Similarly, the “Brevity” problem is related to the previous one, when the documentation fails to provide detailed explanations, featuring overly brief, superficial, and condensed texts that scarcely explain the API’s usage or functionality.

In the category “Presentation”, in addition to the problem identified by Uddin and Robillard [55], we only found a single new problem: “Lack of standardization”. This problem is related to the quality dimension “Consistency” from Ajam et al. [2], considering the failure of following regular usage

of style and visualization across all documentation regarding the presentation of content and format.

The categories “Usability” and “Sustainability” are new, in relation to categories of Uddin and Robillard [55]. “Usability” problems consider how and why the usage of the documentation is hampered by diverse factors. The most recurring problem of this category, is the lack of documentation for a given API. For the “Sustainability” problems, the most recurring problem is the “Compromised Maintenance”, that refers to inconsistency in maintaining the documentation for any reason (e.g. updating the documentation is high-cost, or simply neglected).

4.3 RQ2. What are the main impacts, or consequences, related to the problems raised in RQ1?

We mapped **20 impacts** arising from documentation failures, which were organized into four dimensions:

- **Usability.** Impacts that hinder the use, access, or understanding of API documentation information for the reader;
- **Team.** Impacts that directly affect how teams interact with or consume the API and its documentation, causing socio-technical and productivity losses;
- **Content.** Impacts that degrade the quality or structure of the documentation content itself;
- **Sustainability.** Impacts that jeopardize the long-term longevity and future maintainability of the documentation.

Table 2 maps these impacts to the primary studies from which they were observed, organized by dimensions and ordered by occurrence. The “Usability” and “Team” dimensions concentrate the vast majority of citations, proving that documentation deficiencies directly affect the daily routine, technical efficiency, and psychological factors of the developer.

The top three most recurring impacts belong to the “Usability” dimension: “Impacted understanding” (30 citations), “Impacted usage” (25 citations), and “Unexpected behavior” (14 citations). Conceptually, these three impacts are deeply intertwined within web and distributed environments. A failure in documentation first triggers an “Impacted understanding”, which naturally can cascade into “Impacted usage” when developers attempt to integrate the API. Consequently, this flawed execution can manifest as an “Unexpected behavior” at runtime, leading to integration bottlenecks and software defects.

In the “Team” dimension, the consequences shift from purely technical challenges to socio-technical degradation. The most recurring impacts are “Demotivation” (8 citations), “Third-party information usage” (7 citations), and “API abandonment” (6 citations). There is a clear behavioral progression among these factors: when official artifacts fail, developers are forced to use information from third-party sources, hunting for unverified answers in unofficial forums. The persistent frustration of navigating fragmented data ultimately breeds severe “Demotivation” within software engineering teams,

Table 1: Catalog of problems related to API documentation. Shaded rows indicate new problems in relation to [55].

Category	Problem	Description	Qty	Citations
Content	Incompleteness	The documentation does not provide a complete description of an API element or topic.	41	[1–4, 8, 9, 12, 14, 15, 17–25, 27, 30–38, 40, 43, 47, 49, 51, 52, 54, 57, 61, 63–66]
	Incorrectness	The documentation presents information that is incorrect, inaccurate, or inconsistent with the application's actual operation.	14	[1–3, 8, 18–22, 34, 36, 59, 60, 64]
	<i>Human-only</i>	The documentation does not provide structured resources (such as OpenAPI/Swagger) and is intended solely for human reading.	12	[5–7, 10, 16, 35, 40, 42, 46, 59, 60, 66]
	Obsolescence	The documentation does not provide up-to-date information, referencing earlier versions that are inconsistent with the API's actual behavior.	11	[1, 12, 18, 20, 27, 30, 31, 33, 36, 52, 58]
	Hard comprehension	The documentation uses overly complex language, features non-intuitive descriptions, or requires a high level of prior knowledge.	10	[1, 3, 4, 11, 25, 29, 30, 35, 36, 58]
	Inconsistency	The documentation presents conflicting or contradictory information, or information that is inconsistent across different sections.	9	[2, 3, 11, 22, 23, 34, 58, 64, 65]
	Bias	The documentation does not provide a comprehensive overview; it is constructed from a single perspective and fails to consider the diverse needs of developers.	6	[8, 12, 15, 26, 29, 30]
	Ambiguity	The documentation presents information that is unclear, allowing for multiple interpretations or making it difficult to understand an element.	5	[1, 18, 19, 51, 64]
	Examples without explanation	The documentation presents code examples without explanations or with insufficient details regarding their functioning, purpose, or context.	2	[15, 39]
	Brevity	The documentation lacks detailed explanations, offering texts that are far too short, superficial, and condensed, barely explaining how to use the API or how it works.	1	[62]
Total			111	-
Presentation	Bloat	The documentation features lengthy texts with overblown explanations and an excess of unnecessary descriptions.	9	[1, 8, 18, 25, 32, 36, 39, 58, 62]
	Lack of standardization	The documentation does not exhibit a consistent pattern of structure, writing, organization, or information presentation across the API.	8	[1, 8, 10, 22, 31, 40, 53, 63]
	Fragmentation	The documentation does not present information centrally, scattering related content across different pages or sections.	4	[19, 25, 32, 39]
	Structural information excess	The documentation presents excessive information regarding syntax or structure that can already be easily obtained via IDEs, rendering the content redundant.	3	[1, 10, 25]
	Mixed information	The documentation presents a description of an element mixed with information that is irrelevant or unnecessary for the developer's context.	2	[25, 39]
Total			26	-
Usability	Nonexistence	The documentation does not present any available content for the API, its elements, or its functionalities.	14	[2, 12, 15, 17, 18, 22, 29, 40, 45, 51, 58–60, 66]
	Lack of <i>feedback</i>	The documentation does not provide means or channels for users to submit feedback, report issues, or clarify questions.	3	[19, 30, 63]
	Third-party information usage	The official documentation does not provide sufficient information, forcing the developer to seek answers in forums, blogs, or third-party videos.	3	[18, 36, 41]
	Difficult navigation	The documentation lacks an easy-to-navigate structure, making it difficult to locate information or understand how the content is organized.	2	[17, 66]
	Not followed	Regardless of whether the documentation provides clear guidelines and definitions, developers do not follow or ignore them when using the API.	2	[33, 53]
	Difficult access	The documentation is not freely accessible; it is protected by barriers such as mandatory authentication, restricted permissions, or payment requirements.	2	[1, 40]
	No search function	The documentation does not feature a search mechanism that allows for the quick location of specific information, topics, or elements.	1	[36]
Total			27	-
Sustainability	Compromised maintenance	Documentation maintenance is neglected, costly, or hindered by an API-related factor.	16	[1–4, 8, 11, 18, 41, 45, 48, 56–58, 60, 64, 65]
	Technological limitations	The documentation presents limitations regarding creation, maintenance, or quality due to restrictions imposed by the language, framework, or API technologies.	1	[48]
	<i>Crowdsourced</i>	The documentation lacks a centralized official base, being built or maintained primarily through community contributions.	1	[46]
	Duplicate	The documentation presents the same API, or a portion of it, replicated across multiple locations or versions.	1	[1]
Total			19	-

Table 2: Cataloged impacts

Category	Impact	Description	N	Citations
Usability	Impacted understanding	The problem directly impacts the developers' understanding of the API.	30	[1, 2, 4, 5, 8, 12, 14, 15, 18, 20–22, 22, 25, 33, 35, 36, 39, 40, 45, 46, 51–54, 61, 62, 64–66]
	Impacted usage	The problem hinders the developer from using the API or causes them to use it incorrectly.	25	[1, 2, 4, 8, 12, 16–18, 22, 23, 25, 30–34, 36, 39, 45, 48, 49, 53, 61, 63, 66]
	Unexpected behavior	The problem causes the API, or the software using it, to present errors or unexpected behavior.	14	[2, 17, 20, 21, 23, 32–34, 40, 49, 52, 53, 64, 65]
	Impacted tool usage	The problem impacts the usage or creation of tools for API documentation and recommendations.	9	[22, 31, 40, 42, 48, 58–60, 62]
	Nonexistence	The problem causes the documentation to be nonexistent.	3	[1, 45, 66]
	Impacted documentation access	The problem causes developers to have compromised access to the documentation in some way.	1	[1]
Total			82	
Team	Demotivation	The problem causes developers to feel demotivated to use the API, lose trust in it, or become frustrated.	8	[1, 5, 8, 11, 31, 36, 64, 65]
	Third-party information seeking	The problem causes developers to look for information in unofficial places.	7	[19, 24, 36, 39, 43, 52, 65]
	API abandonment	The problem causes developers to abandon the API and likely seek another one.	6	[2, 39, 63–66]
	Impacted API visibility	The problem tarnishes the API's image or makes it difficult for developers to find it.	4	[7, 9, 14, 36]
	Impacted client software	The issue impacts software that uses the documented API.	4	[31, 41, 64, 65]
	Time wasting	The problem causes developers using the API to waste time or experience a decrease in productivity.	2	[24, 64]
	Ignored documentation	The problem causes the documentation to be intentionally ignored.	1	[53]
Total			32	
Content	Obsolescence	The problem causes the documentation to become outdated.	2	[56, 57]
	Ambiguity	The problem causes ambiguity in the documentation.	2	[5, 31]
	Bloat	The problem causes the documentation to become bloated.	2	[8, 46]
	Incompleteness	The problem causes the documentation to become incomplete.	1	[57]
	Lack of standardization	The problem causes a lack of standardization in the documentation.	1	[40]
	Incorrectness	The problem causes the documentation to become incorrect.	1	[65]
Total			9	
Sustainability	Compromised maintenance	The problem brings difficulties for the maintenance of the documentation to occur.	10	[2, 8, 25, 30, 42, 47, 48, 59, 63, 66]
Total			10	

which frequently can result in a critical business impact: the outright “API abandonment” in favor of a better-documented competitor.

Conversely, the “Content” and “Sustainability” dimensions act as internal systemic degradation factors. For “Content”, the impacts are sparsely but evenly distributed among “Obsolescence” (2 citations), “Ambiguity” (2 citations), and “Bloat” (2 citations). A notable **cascade effect** was identified here: these impacts correlate directly with problems of the same

nature raised in RQ1, indicating that an initial authoring failure tends to feed back into the lifecycle, generating new negative content developments over time.

Finally, the “Sustainability” category reflects how current documentation issues compromise future engineering health. The only impact cataloged in this category is “Compromised maintenance” (10 citations). This high volume of citations highlights that documentation decay is not a static issue;

rather, it threatens the technical debt of the system, signaling that once documentation begins to fail, its ongoing maintenance becomes exponentially high-cost or completely neglected by the development team.

4.4 RQ3. What are the actions proposed in the literature to solve these documentation problems?

The analysis identified **8 proposed actions**, in the literature, to mitigate and resolve the mapped technical deficiencies. These solutions range across different operational natures, encompassing automation-driven efforts focused on computational tools, process-driven approaches aimed at introducing internal organizational workflows, and community-driven strategies that leverage collaborative knowledge. Table 3 maps these actions to the primary studies from which they were observed, detailed by their descriptions, hierarchical “Parent” relationships, total occurrence (N), and corresponding citations.

The solutions focus heavily on two predominant, high-level approaches: “Use tool” (31 citations) and “Improve documentation” (22 citations). Conceptually, the massive concentration of research in “Use tool” highlights a strong consensus in academia toward automation as the primary mechanism to solve documentation failures. This implies that manually addressing documentation quality is increasingly viewed as unfeasible or counterproductive in modern software development. Instead, the literature heavily invests in engineering automated tools to minimize human error during documentation authoring and maintenance.

A distinguishing architectural characteristic of the proposed solutions is the hierarchy where specific actions derive from broader “Parent” concepts. The foundational action “Improve documentation” serves as a generalization for three specialized, process-driven sub-actions: “Documentation formatting” (4 citations), “Mandatory documentation update” (1 citation), and “Adopt a standard location” (1 citation).

Among these specialized branches, “Documentation formatting” stands out by advocating for the strict adoption of specifications like OpenAPI. This indicates that a significant subset of process-driven research aims to combat problems like inconsistency and lack of standardization by enforcing machine-readable, structured templates. Conversely, the extremely low occurrence of “Mandatory documentation update” and “Adopt a standard location” reveals a critical gap: while the literature acknowledges that documentation must be improved, it rarely proposes concrete, formalized workflows to embed these improvements into the daily DevOps or agile routines of development teams.

Except for the two predominant approaches, all other actions were cited by fewer than 5 works. For instance, collaborative or feedback-loop mechanisms such as “Enable feedback” (2 citations) and “Consult the community” (2 citations) remain highly underrepresented. This strong polarization evidences that the literature is still in an early stage regarding the exploration of socio-technical or decentralized

solutions. The current state of the art relies almost exclusively on immediate automation or generic manual improvements, revealing a clear scientific gap for the formulation of innovative methodological, interactive, or conceptual approaches to solve the lifecycle challenges of API documentation.

5 DISCUSSION

This section discusses the implications of the SMS findings, structured into five axes derived from the most significant patterns identified in Sections 4.2–4.4: **(1)** the expansion of the scope beyond content/presentation captures the new reality of documentation as ecosystem artifact (Section 5.1); **(2)** the interconnection among problems, impacts, and actions reveals systemic patterns (Section 5.2); **(3)** the polarization of proposed actions toward automation signals a gap in organizational approaches (Section 5.3); **(4)** the emergence of machine-readable documentation reflects contemporary Web demands (Section 5.4); and **(5)** the interdisciplinarity confirms that documentation quality requires cross-field solutions (Section 5.5).

5.1 Documentation Problems vs. Documentation-Related Problems

Differing from previous static taxonomies that evaluated the documentation artifact in isolation under the lenses of Content and Presentation [55], this study expands the scope to the ecosystem in which the documentation resides. The introduction of the “Usability” and “Sustainability” dimensions allowed for the cataloging of operational and contextual obstacles faced by developers when consuming and maintaining such artifacts over time.

This expansion is justified by the transformations in the software development lifecycle. The gap regarding problems of this nature in past literature raises a question: were these challenges non-existent or underestimated at the time? It is plausible that the continuous evolution of *frameworks*, continuous delivery, and Artificial Intelligence has introduced new classes of ecological problems that were previously irrelevant. In parallel, while tool-focused studies like the *Scout-bot* [2, 31] prioritize the quantitative measurement of quality metrics (completeness and consistency), this work focuses on the “Problem-Impact-Action” causal triad, offering complementary insights oriented toward interventions.

Furthermore, the absence of the “Nonexistence” problem in the Uddin and Robillard [55] work is noteworthy, contrasting with this mapping. This suggests that contemporary literature presupposes a baseline level of documentation existence for public APIs, strictly focusing its efforts on the internal quality and evolution of the already written artifact. Thus, the taxonomic contribution of this study consolidates a dynamic and contextualized view, essential for subsidizing future research in Software Engineering.

Table 3: Cataloged actions

Action	Description	Parent	N	Citations
Use tool	Use a tool to automate, in any way, some documentation process.		31	[1–6, 9, 10, 12, 15, 22, 23, 26, 29–31, 33, 35, 38, 40, 43, 45, 48, 51, 53, 56, 57, 59–61, 66]
Improve documentation	Create documentation or adopt actions to improve existing documentation.		22	[1–3, 7, 8, 11, 12, 14, 17, 19, 22, 27, 31, 33, 34, 42, 47–49, 51, 63, 66]
Documentation formatting	Adopt a specific documentation formatting standard, such as OpenAPI.	Improve documentation	4	[26, 59, 60, 63]
Enable feedback	Provide a medium for developers to give feedback regarding the documentation.		2	[2, 63]
Consult the community	Leverage information that has already been produced by the developer community.		2	[52, 61]
Mandatory documentation update	Establish mandatory documentation maintenance within the team's development processes.	Improve documentation	1	[1]
Adopt a standard location	Adopt a standardized location to host and store documentations.	Improve documentation	1	[1]
Seek third-party information	Look for information in external or unofficial places.		1	[1]

5.2 Relationship between Problems, Impacts and Actions

The Sankey diagram ⁴ (Figure 6) illustrates the mapped interdependence and causality among Impacts (left column; standard “I00”), Problems (central column; standard “P00”), and Proposed Actions (right column; standard “AC00”). The high density of connections stresses that documentation deficiencies constitute multifaceted phenomena with severe repercussions on the efficiency of software teams.

Structurally, central failures such as “Incompleteness”, “Inconsistency”, and “Compromised Maintenance” act as highly connected hubs and root causes, radiating bulky branches into the Impacts dimension. On the other hand, the graph exposes a strategic misalignment between causes and solutions: the literature concentrates predominantly on actions that are easy to implement technically, such as automated generation and validation tools, neglecting directed interventions to interrupt the propagation chain of the highest-impact problems. The diagram further exposes a cyclical feedback loop of failures, where certain impacts (e.g., schedule delays) turn into catalysts for new problems (e.g., documentation obsolescence), demanding systemic mitigation approaches.

5.3 Lack of Variety in Proposed Actions

The analysis of the proposed solutions points to a severe homogeneity in the literature, which is heavily polarized toward the development of automated software artifacts to

the detriment of social or organizational approaches, such as “Consult the community” or “Enable feedback channels”.

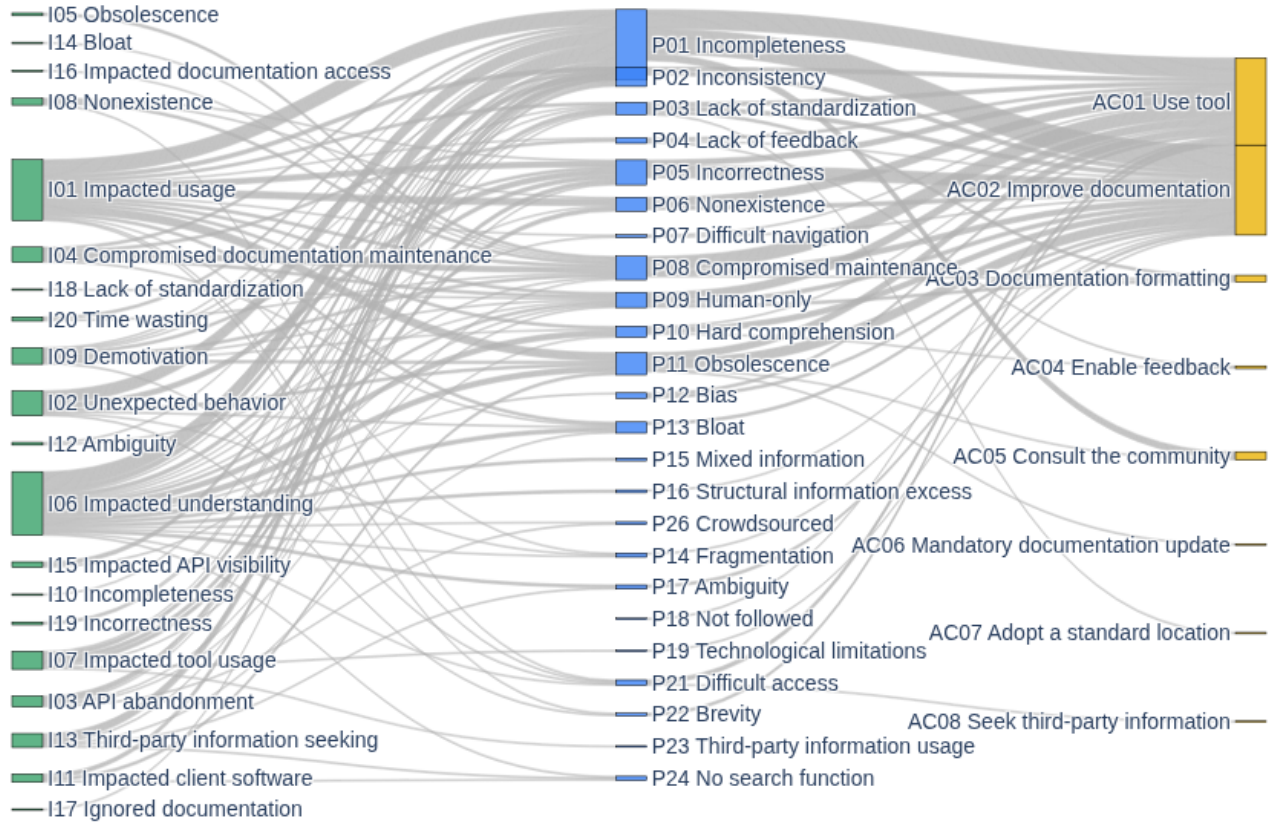
This concentration on automated solutions reflects the ease of measuring software metrics and the speed of publishing tangible artifacts. However, this asymmetry overlooks the human and governance dimensions necessary for documentation sustainability. Mitigating failures like “Incompleteness” or “Inconsistency” requires efficient channels for bimodal collaboration between API producers and consumers. By focusing excessively on syntax and structural automation, research neglects the contextual semantics of use, limiting practical options for the software industry.

5.4 Made for Humans vs. Made for Machines

The detection of the “*Human-only*” problem, documentations designed exclusively for human reading and lacking structured, machine-readable formats, reflects contemporary automation demands. If purely textual manuals were once tolerated, the proliferation of web ecosystems and automated integrations has turned formats like OpenAPI/Swagger into critical non-functional requirements.

Currently, documentations operate as primary structured inputs for Software Engineering tools, including AI code generators and Large Language Models (LLMs). However, the heavy reliance on data generated by humans without technical validation introduces cybersecurity risks. As evidenced by Kudriavtseva et al. [28], code generators trained on open repositories containing vulnerabilities tend to replicate insecure patterns. Analogously, documentations based on informal community knowledge (*crowdsourced*) propagate inaccuracies and suboptimal examples that are subsequently

⁴Interactive version available at <https://alligware-ops.github.io/sankey/>

Figure 6: Correlations among Impacts, Problems, and Actions

absorbed by automations. The transition to a format that is simultaneously human- and machine-readable therefore demands design tools that ensure rigorous alignment between code, documentation, and executable contracts.

5.5 Related Computing Areas

To conduct the documentation research and analysis cataloged in this study, the investigated papers integrated multiple subfields of Computer Science. The use of Machine Learning [19, 47, 52] enabled the automatic classification of relevant textual fragments; Natural Language Processing (NLP) techniques [9, 33, 34, 52, 63, 65] allowed for data extraction from unstructured documentations; and Data Mining techniques [4, 9, 25, 30, 31, 33, 40, 45, 63] made it possible to mine massive Q&A repositories from platforms like *Stack Overflow*. This interdisciplinarity confirms that improving API documentation quality constitutes a cross-disciplinary challenge, capable of mobilizing and benefiting from multiple scientific specialties.

6 LIMITATIONS

This work proposes a systematic mapping study to capture the academic community's ongoing debates regarding API

documentation problems. However, it is subject to certain limitations.

First, this study utilizes only the Scopus database to retrieve relevant papers. Although Scopus indexes major publishers (ACM, IEEE, Springer), direct searches in other databases, such as ACM Digital Library and IEEE Xplore, could potentially retrieve studies not captured by Scopus's indexing. This single-source constraint introduces a risk of missing important studies. To mitigate this limitation, the snowballing technique was applied, but it does not fully substitute multi-database searches as recommended by SMS guidelines [44]. Additionally, the exclusion of studies based on institutional access limitations (EC04) may have introduced availability bias, potentially not including relevant work behind paywalls not covered by our institutional subscriptions.

Second, there is a possibility that the mapping, despite the execution of the search *string* and the subsequent *snowballing* process, did not encompass all articles that addressed or cited issues related to API documentation problems. Consequently, the selected corpus might represent only a subset of a broader group of studies in the field.

Third, another factor to consider is that the included articles and the data extracted in this mapping are directly

subject to the researchers' interpretation. This subjectivity may have introduced a certain degree of researcher bias during the application of the inclusion and exclusion criteria, as well as during the information extraction and categorization phases.

7 CONCLUSIONS

This systematic mapping study synthesizes the academic debate surrounding API documentation failures between 2016 and 2025. From the 61 primary studies analyzed, 26 problems were cataloged, confirming and updating the baseline study "How API Documentation Fails" [55] both with new problems within its previously identified categories and with two new categories, "Usability" and "Sustainability". This indicates that both historical and newly emerging problems continue to persist in modern development. We additionally mapped 20 impacts and 8 mitigating actions related to the identified documentation-related problems.

Furthermore, this study demonstrates that these problems affect multiple dimensions of the relationship between an API and its consumers: documentation deficiencies trigger severe negative impacts on how programmers utilize, comprehend, and even emotionally perceive an API. Inefficient documentation often leads to integration bottlenecks, project delays, developer demotivation, or the outright abandonment of a technology. Within the domain of Web systems, applications are increasingly architected as compositions of REST APIs and microservices, and machine-readable specifications such as OpenAPI/Swagger serve as the primary contract between providers and consumers. Moreover, Web APIs play a central role in contemporary multimedia and hypermedia applications. Therefore, API documentation quality emerges as a critical factor for the modern Web ecosystem, as it often determines whether a developer or a software engineering team will successfully adopt and compose distributed Web services. Our findings reinforce the need for the Web development community to adopt higher-quality documentation practices.

However, despite the expanding catalog of problems, innovations within the field were also identified. The most prominent of these involves the deployment of automated tools, whose approaches range from generating documentation directly from code to synthesizing technical examples or restructuring legacy documentation. These automated strategies effectively relieve developers of onerous manual documentation tasks.

This study opens a wide range of opportunities for future research. The immediate next step involves validating the empirical prevalence of the discovered problems within the daily routines of developers in the software industry. Additionally, we aim to gather the impacts experienced and the mitigation actions proposed by industry professionals. This field data will enable us to draw a parallel between the challenges documented in the literature and those faced in the market regarding API documentation.

Beyond data collection, we intend to develop a technical guide to compile all identified problems, impacts, and

proposed actions in a didactic manner. This artifact will serve to raise awareness and guide developers who either consume or construct API documentation, thereby promoting a deeper understanding of the importance of high-quality documentation.

8 ETHICAL CONSIDERATIONS

This work is a secondary study based exclusively on the analysis of previously published academic literature. No human subjects participated in any stage, no personal or sensitive data were collected, and no experiments involving people were conducted. The analyzed corpus consists solely of bibliographic records and full texts of publicly available scientific papers, retrieved through institutional subscriptions (CAPES portal) in compliance with the publishers' licenses; no anonymization was required, since no personal data were processed. Under Resolution CNS No. 510/2016 of the Brazilian CEP/CONEP system, research based exclusively on publicly accessible information is exempt from review by a research ethics committee, and therefore no CAAE registration applies to this study.

ACKNOWLEDGMENTS

In accordance with the SBC Code of Conduct for Authors, the authors declare the use of Google Gemini 3.5 Flash, and that the tool usage was strictly limited to grammar review and text refinement. The AI was not used to generate data, graphs, or citations, nor is it credited as a co-author. The human authors bear full ethical, scientific, and originality responsibility for the article's content.

REFERENCES

- [1] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software Documentation Issues Unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 1199–1210. doi:10.1109/ICSE.2019.00122
- [2] George Ajam, Carlos Rodriguez, and Boualem Benatallah. 2021. Scout-bot: Leveraging API Community Knowledge for Exploration and Discovery of API Learning Resources. *CLEI Electronic Journal* 24 (07 2021). doi:10.19153/cleiej.24.2.5
- [3] George Ajam, Carlos Rodríguez, and Boualem Benatallah. 2020. API Topics Issues in Stack Overflow QAs Posts: An Empirical Study. In *2020 XLVI Latin American Computing Conference (CLEI)*. 147–155. doi:10.1109/CLEI52000.2020.00024
- [4] Seham Alharbi, Dimitris Kolovos, and Nicholas Matragkas. 2022. Synthesising Linear API Usage Examples for API Documentation. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 607–611. doi:10.1109/ICSME55016.2022.00084
- [5] Leonardo Araujo, Guillermo Rodríguez, Santiago Vidal, Claudia Marcos, and Rodrigo dos Santos. 2022. Empirical Analysis on OpenAPI Topic Exploration and Discovery to Support the Developer Community. *Computing and Informatics* 40 (02 2022), 1345–1369. doi:10.31577/cai_2021_6_1345
- [6] Mehdi Bahrami and Wei-Peng Chen. 2019. WATAPI: Composing Web API Specification from API Documentations through an Intelligent and Interactive Annotation Tool. In *2019 IEEE International Conference on Big Data (Big Data)*. 4573–4578. doi:10.1109/BigData47090.2019.9006355
- [7] Mehdi Bahrami and Wei-Peng Chen. 2020. Automated Web Service Specification Generation Through a Transformation-Based Learning. In *Services Computing – SCC 2020*, Qingyang Wang, Yunni Xia, Sangeetha Seshadri, and Liang-Jie Zhang (Eds.). Springer International Publishing, Cham, 103–119.
- [8] Gloria Bondel, Arif Cerit, and Florian Matthes. 2022. Challenges of API Documentation from a Provider Perspective and Best

- Practices for Examples in Public Web API Documentation. In *Proceedings of the 24th International Conference on Enterprise Information Systems - Volume 2: ICEIS*. INSTICC, SciTePress, 268–279. doi:10.5220/0011089700003179
- [9] Liang Cai, Haoye Wang, Qiao Huang, Xin Xia, Zhenchang Xing, and David Lo. 2019. BIKER: a tool for Bi-information source based API method recommendation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) (*ESEC/FSE 2019*). Association for Computing Machinery, New York, NY, USA, 1075–1079. doi:10.1145/3338906.3341174
 - [10] Hanyang Cao, Jean-Rémy Falleri, and Xavier Blanc. 2017. Automated Generation of REST API Specification from Plain HTML Documentation. In *Service-Oriented Computing: 15th International Conference, ICSOC 2017, Malaga, Spain, November 13–16, 2017, Proceedings* (Malaga, Spain). Springer-Verlag, Berlin, Heidelberg, 453–461. doi:10.1007/978-3-319-69035-3_32
 - [11] Luca Chiodini, Simone Piatti, and Matthias Hauswirth. 2024. Judicious: API Documentation for Novices. In *Proceedings of the 2024 ACM SIGPLAN International Symposium on SPLASH-E* (Pasadena, CA, USA) (*SPLASH-E '24*). Association for Computing Machinery, New York, NY, USA, 89–97. doi:10.1145/3689493.3689987
 - [12] Michael Coblenz, William Guo, Kevin Voozhian, and Jeffrey S. Foster. 2023. A Qualitative Study of REST API Design and Specification Practices. In *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 148–156. doi:10.1109/VL-HCC57772.2023.00025
 - [13] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, 275–284.
 - [14] Alex Cummaudo, Rajesh Vasa, John Grundy, and Mohamed Abdelrazek. 2022. Requirements of API Documentation: A Case Study into Computer Vision Services. *IEEE Transactions on Software Engineering* 48, 6 (2022), 2010–2027. doi:10.1109/TSE.2020.3047088
 - [15] Fernanda Madeiral Delfim, Klérissou V. R. Paixão, Damien Cas-sou, and Marcelo de Almeida Maia. 2016. Redocumenting APIs with crowd knowledge: a coverage analysis based on question types. *Journal of the Brazilian Computer Society* 22, 1 (2016), 9. doi:10.1186/s13173-016-0049-0
 - [16] Hamza Ed-doubi, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2017. Example-Driven Web API Specification Discovery. In *Modelling Foundations and Applications*, Anthony Anjorin and Huáscar Espinoza (Eds.). Springer International Publishing, Cham, 267–284.
 - [17] Akalanka Galappaththi, Sarah Nadi, and Christoph Treude. 2024. An Empirical Study of API Misuses of Data-Centric Libraries. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Barcelona, Spain) (*ESEM '24*). Association for Computing Machinery, New York, NY, USA, 245–256. doi:10.1145/3674805.3686685
 - [18] Andrew Head, Caitlin Sadowski, Emerson Murphy-Hill, and Andrea Knight. 2018. When not to comment: questions and tradeoffs with API documentation for C++ projects. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) (*ICSE '18*). Association for Computing Machinery, New York, NY, USA, 643–653. doi:10.1145/3180155.3180176
 - [19] Amber Horvath, Michael Xieyang Liu, River Hendriksen, Connor Shannon, Emma Paterson, Kazi Jawad, Andrew Macvean, and Brad A Myers. 2022. Understanding How Programmers Can Use Annotations on Documentation (*CHI '22*). Association for Computing Machinery, New York, NY, USA, Article 69, 16 pages. doi:10.1145/3491102.3502095
 - [20] Masaki Hosono, Hironori Washizaki, Yoshiaki Fukazawa, and Kiyoshi Honda. 2018. An Empirical Study on the Reliability of the Web API Document. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. 715–716. doi:10.1109/APSEC.2018.00103
 - [21] Masaki Hosono, Hironori Washizaki, Kiyoshi Honda, Hiromasa Nagumo, Hisanobu Sonoda, Yoshiaki Fukazawa, Kazuki Munakata, Takao Nakagawa, Yusuke Nemoto, Susumu Tokumoto, and Supast Monpratarnchai. 2019. Inappropriate Usage Examples in Web API Documentations. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 343–347. doi:10.1109/ICSME.2019.00052
 - [22] Peiwei Hu, Ruigang Liang, Ying Cao, Kai Chen, and Runze Zhang. 2023. AURC: detecting errors in program code and documentation. In *Proceedings of the 32nd USENIX Conference on Security Symposium* (Anaheim, CA, USA) (*SEC '23*). USENIX Association, USA, Article 80, 18 pages.
 - [23] Ruikai Huang, Manish Motwani, Idel Martinez, and Alessandro Orso. 2024. Generating REST API Specifications through Static Analysis. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) (*ICSE '24*). Association for Computing Machinery, New York, NY, USA, Article 107, 13 pages. doi:10.1145/3597503.3639137
 - [24] Siyuan Jiang, Ameer Armaly, Collin McMillan, Qiyu Zhi, and Ronald Metoyer. 2018. Documenting API Input/Output Examples. arXiv:1703.09613 [cs.SE] <https://arxiv.org/abs/1703.09613>
 - [25] Junaed Younus Khan, Md. Tawkat Islam Khondaker, Gias Uddin, and Anindya Iqbal. 2021. Automatic Detection of Five API Documentation Smells: Practitioners' Perspectives. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 318–329. doi:10.1109/SANER50967.2021.00037
 - [26] István Koren and Ralf Klamma. 2018. The Exploitation of OpenAPI Documentation for the Generation of Web Frontends. In *Companion Proceedings of the The Web Conference 2018* (Lyon, France) (*WWW '18*). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 781–787. doi:10.1145/3184558.3188740
 - [27] Reditana Koçi, Xavier Franch, Petar Jovanovic, and Alberto Abelló. 2019. Classification of Changes in API Evolution. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*. 243–249. doi:10.1109/EDOC.2019.00037
 - [28] Arina Kudriavtseva, Nisar Ahmad Hotak, and Olga Gadyatskaya. 2025. My Code Is Less Secure with Gen AI: Surveying Developers' Perceptions of the Impact of Code Generation Tools on Security. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing* (Catania International Airport, Catania, Italy) (*SAC '25*). Association for Computing Machinery, New York, NY, USA, 1637–1646. doi:10.1145/3672608.3707778
 - [29] Raquel Lafeta, Thiago Lafeta, and Marcelo Maia. 2020. An Automated Approach for Constructing Framework Instantiation Documentation. *International Journal of Software Engineering and Knowledge Engineering* 30 (04 2020), 575–601. doi:10.1142/S0218194020500205
 - [30] Maxime Lamothe. 2020. Bridging the Divide Between API Users and API Developers by Mining Public Code Repositories. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 178–181.
 - [31] Seonah Lee, Rongxin Wu, Shing-Chi Cheung, and Sungwon Kang. 2021. Automatic Detection and Update Suggestion for Outdated API Names in Documentation. *IEEE Transactions on Software Engineering* 47, 4 (2021), 653–675. doi:10.1109/TSE.2019.2901459
 - [32] Hongwei Li, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, and Xuejiao Zhao. 2018. Improving API Caveats Accessibility by Mining API Caveats Knowledge Graph. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 183–193. doi:10.1109/ICSME.2018.00028
 - [33] Li Li, Jun Gao, Tegawendé F. Bissyandé, Lei Ma, Xin Xia, and Jacques Klein. 2018. Characterising deprecated Android APIs (*MSR '18*). Association for Computing Machinery, New York, NY, USA, 254–264. doi:10.1145/3196398.3196419
 - [34] Ruishi Li, Yunfei Yang, Jinghua Liu, Peiwei Hu, and Guozhu Meng. 2022. The inconsistency of documentation: a study of online C standard library documents. *Cybersecurity* 5 (07 2022), 14. doi:10.1186/s42400-022-00118-9
 - [35] Meherun Nesa Lucky, Marco Cremaschi, Barbara Lodigiani, Antonio Menolascina, and Flavio De Paoli. 2016. Enriching API Descriptions by Adding API Profiles Through Semantic Annotation. In *Service-Oriented Computing*, Quan Z. Sheng, Eleni Stroulia, Samir Tata, and Sami Bhiri (Eds.). Springer International Publishing, Cham, 780–794.
 - [36] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. 2018. Application Programming Interface Documentation: What Do Software Developers Want? *Journal of Technical Writing and Communication* 48 (07 2018), 295–330. doi:10.1177/0047281617721853
 - [37] Michael Meng, Stephanie M. Steinhardt, and Andreas Schubert. 2020. Optimizing API Documentation: Some Guidelines and Effects. In *Proceedings of the 38th ACM International Conference*

- on *Design of Communication* (Denton, TX, USA) (*SIGDOC '20*). Association for Computing Machinery, New York, NY, USA, Article 24, 11 pages. doi:10.1145/3380851.3416759
- [38] Michał Michalski, Piotr Kosko, Dawid Juszcak, and Hobum Kwon. 2020. EWIDL: Single-Source Web API Documentation Management System. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 723–726. doi:10.1109/ICSME46990.2020.00081
- [39] Mathieu Nassif, Zara Horiacher, and Martin P. Robillard. 2022. Casdoc: Unobtrusive Explanations in Code Examples. In *2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC)*. 631–635. doi:10.1145/3524610.3527730
- [40] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. 2021. An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing* 14, 4 (2021), 957–970. doi:10.1109/TSC.2018.2847344
- [41] Kristian Nybom, Adnan Ashraf, and Ivan Porres. 2018. A Systematic Mapping Study on API Documentation Generation Approaches. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 462–469. doi:10.1109/SEAA.2018.00081
- [42] Rahul Pandita, Kunal Taneja, Laurie Williams, and Teresa Tung. 2016. ICON: Inferring Temporal Constraints from Natural Language API Descriptions. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 378–388. doi:10.1109/ICSME.2016.59
- [43] Xin Peng, Yifan Zhao, Mingwei Liu, Fengyi Zhang, Yang Liu, Xin Wang, and Zhenchang Xing. 2018. Automatic Generation of API Documentations for Open-Source Projects. In *2018 IEEE Third International Workshop on Dynamic Software Documentation (DySDoc3)*. 7–8. doi:10.1109/DySDoc3.2018.00010
- [44] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic mapping studies in software engineering. In *12th international conference on evaluation and assessment in software engineering (EASE)*. BCS Learning & Development.
- [45] Hung Phan, Hoan Anh Nguyen, Tien N. Nguyen, and Hridesh Rajan. 2017. Statistical Learning for Inference between Implementations and Documentation. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*. 27–30. doi:10.1109/ICSE-NIER.2017.9
- [46] Martin P. Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez, Gail C. Murphy, Laura Moreno, David Shepherd, and Edmund Wong. 2017. On-demand Developer Documentation. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 479–483. doi:10.1109/ICSME.2017.17
- [47] Anand Ashok Sawant, Guangzhe Huang, Gabriel Vilen, Stefan Stojkovski, and Alberto Bacchelli. 2018. Why are Features Deprecated? An Investigation Into the Motivation Behind Deprecation. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 13–24. doi:10.1109/ICSME.2018.00011
- [48] S. M. Sohan, Craig Anslow, and Frank Maurer. 2017. Automated example oriented REST API documentation at Cisco. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. 213–222. doi:10.1109/ICSE-SEIP.2017.11
- [49] S M Sohan, Frank Maurer, Craig Anslow, and Martin P. Robillard. 2017. A study of the effectiveness of usage examples in REST API documentation. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 53–61. doi:10.1109/VLHCC.2017.8103450
- [50] Donna Spencer. 2009. *Card sorting: Designing usable categories*. Rosenfeld Media.
- [51] Peng Sun, Chris Brown, Ivan Beschastnikh, and Kathryn T. Stolee. 2019. Mining Specifications from Documentation using a Crowd. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 275–286. doi:10.1109/SANER.2019.8668025
- [52] Yida Tao, Jiefang Jiang, Yepang Liu, Zhiwu Xu, and Shengchao Qin. 2021. Understanding performance concerns in the API documentation of data science libraries. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (Virtual Event, Australia) (ASE '20)*. Association for Computing Machinery, New York, NY, USA, 895–906. doi:10.1145/3324884.3416543
- [53] Yuan Tian, Ferdian Thung, Abhishek Sharma, and David Lo. 2017. APiBot: Question answering bot for API documentation. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 153–158. doi:10.1109/ASE.2017.8115628
- [54] Gias Uddin, Foutse Khomh, and Chanchal K Roy. 2019. Towards Crowd-Sourced API Documentation. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 310–311. doi:10.1109/ICSE-Companion.2019.00129
- [55] Gias Uddin and Martin P. Robillard. 2015. How API Documentation Fails. *IEEE Software* 32, 4 (July 2015), 68–75. doi:10.1109/MS.2014.80. Disponível em: <https://doi.org/10.1109/MS.2014.80>. Acesso em: 28 maio 2025.
- [56] Daria Usova. 2023. Generating a Formalized REST API Definition Language for a Dynamic Framework Without Annotations Using Static Analysis Tools. In *Proceedings of the 2023 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. 1710–1714. doi:10.1109/EDM58354.2023.10225100. Disponível em: <https://doi.org/10.1109/EDM58354.2023.10225100>. Acesso em: 28 maio 2025.
- [57] Shujun Wang, Yongqiang Tian, and Dengcheng He. 2023. gDoc: Automatic Generation of Structured API Documentation. In *Companion Proceedings of the ACM Web Conference 2023* (Austin, TX, USA) (*WWW '23 Companion*). Association for Computing Machinery, New York, NY, USA, 53–56. doi:10.1145/3543873.3587310
- [58] Rahulkrishna Yandrapally, Saurabh Sinha, Rachel Tzoref-Brill, and Ali Mesbah. 2023. Carving UI Tests to Generate API Tests and API Specification. arXiv:2305.14692 [cs.SE] <https://arxiv.org/abs/2305.14692>
- [59] Jinqu Yang, Erik Wittern, Annie T. T. Ying, Julian Dolby, and Lin Tan. 2018. Automatically Extracting Web API Specifications from HTML Documentation. arXiv:1801.08928 [cs.SE] <https://arxiv.org/abs/1801.08928>
- [60] Jinqu Yang, Erik Wittern, Annie T. T. Ying, Julian Dolby, and Lin Tan. 2018. Towards extracting web API specifications from documentation. In *Proceedings of the 15th International Conference on Mining Software Repositories (Gothenburg, Sweden) (MSR '18)*. Association for Computing Machinery, New York, NY, USA, 454–464. doi:10.1145/3196398.3196411
- [61] Jingxuan Zhang, He Jiang, Zhilei Ren, Tao Zhang, and Zhiqiu Huang. 2021. Enriching API Documentation with Code Samples and Usage Scenarios from Crowd Knowledge. *IEEE Transactions on Software Engineering* 47, 6 (2021), 1299–1314. doi:10.1109/TSE.2019.2919304
- [62] Neng Zhang, Jian Wang, Keqing He, Zheng Li, and Yiwang Huang. 2019. Mining and clustering service goals for RESTful service discovery. *Knowl. Inf. Syst.* 58, 3 (March 2019), 669–700. doi:10.1007/s10115-018-1171-4
- [63] Neng Zhang, Ying Zou, Xin Xia, Qiao Huang, David Lo, and Shanping Li. 2023. Web APIs: Features, Issues, and Expectations – A Large-Scale Empirical Study of Web APIs From Two Publicly Accessible Registries Using Stack Overflow and a User Survey. *IEEE Transactions on Software Engineering* 49, 2 (2023), 498–528. doi:10.1109/TSE.2022.3154769
- [64] Yu Zhou, Ruihang Gu, Taolue Chen, Zhiqiu Huang, Sebastiano Panichella, and Harald Gall. 2017. Analyzing APIs Documentation and Code to Detect Directive Defects. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 27–37. doi:10.1109/ICSE.2017.11
- [65] Yu Zhou, Changzhi Wang, Xin Yan, Taolue Chen, Sebastiano Panichella, and Harald Gall. 2020. Automatic Detection and Repair Recommendation of Directive Defects in Java API Documentation. *IEEE Transactions on Software Engineering* 46, 9 (2020), 1004–1023. doi:10.1109/TSE.2018.2872971
- [66] Adam Štěpánek, David Kuťák, Barbora Kozlíková, and Jan Byška. 2024. Interactive Diagrams for Software Documentation. In *2024 IEEE Working Conference on Software Visualization (VIS-SOFT)*. 12–23. doi:10.1109/VIS-SOFT64034.2024.00012