Software Documentation: The Practitioners' Perspective

Emad Aghajani[1], Csaba Nagy[1], Mario Linares-Vásquez[2], Laura Moreno[3], Gabriele Bavota[1], Michele Lanza[1], David C. Shepherd[4]

1: REVEAL @ Software Institute, USI Università della Svizzera italiana - Lugano, Switzerland

2: Systems and Computing Engineering Department, Universidad de los Andes, Colombia

3: Department of Computer Science, Colorado State University, USA 4: Virginia Commonwealth University, USA

ABSTRACT

In theory, (good) documentation is an invaluable asset to any software project, as it helps stakeholders to use, understand, maintain, and evolve a system. In practice, however, documentation is generally affected by numerous shortcomings and issues, such as insufficient and inadequate content and obsolete, ambiguous information. To counter this, researchers are investigating the development of advanced recommender systems that automatically suggest high-quality documentation, useful for a given task. A crucial first step is to understand what quality means *for practitioners* and what information is actually needed for specific tasks.

We present two surveys performed with 146 practitioners to investigate (i) the documentation issues they perceive as more relevant together with solutions they apply when these issues arise; and (ii) the types of documentation considered as important given specific tasks. Our findings can help researchers in designing the next generation of documentation recommender systems.

CCS CONCEPTS

• **Software and its engineering → Documentation**.

KEYWORDS

Documentation, Empirical Study

1 INTRODUCTION

A "software post-development issue" [13]. An after-thought, so to speak. This is how the ACM Computing Classification Systems (CCS) categorizes software documentation. Although peculiar, this classification aligns well with the general perception that there are more exciting things to do than documenting software, especially if said software has already been developed.

Studies abound about software documentation being affected by insufficient and inadequate content [1, 54, 55, 69], obsolete and ambiguous information [1, 69, 71], and incorrect and unexplained examples [1, 69], to name just a few issues. In contrast to this rather sad *status quo*, not only are there studies that attest that documentation is actually useful [8, 10, 14, 16, 55], but also there is that thing called "common sense": it simply makes sense to document software—it is just not an activity enjoyed by many.

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00

https://doi.org/10.1145/3377811.3380405

Recent research initiatives [6, 56] have advocated for the development of automated context-aware recommender systems that *automatically generate high-quality documentation*, contextual to any given *task* at hand. This has led to a first wave of automated approaches for the generation and recommendation of documentation (*e.g.,* [23, 40, 44, 50, 57]). While the creation of such novel systems entails conceptual and technical challenges related to the collection, inference, interpretation, selection, and presentation of useful information, it also requires solid empirical foundations on *what* information is (or is not) useful *when* to developers.

To provide such foundations, we [1] recently performed a study to distill a large taxonomy of software documentation issues, and inferred a series of proposals for researchers and practitioners. While our taxonomy was promising, it had not been validated by practitioners, making it mostly an academic construction without the much needed reality check. In this work, our goal is to juxtapose our taxonomy with the documentation needs and priorities of practitioners. The first contribution is thus an empirical validation of the taxonomy to answer our first research question (RQ):

RQ_1: *What documentation issues are relevant to practitioners?*

Previous studies on documentation that were run using surveys with developers focused either on specific issues, *e.g.,* using and learning APIs [54, 55, 69], or were geared towards *generic* activities, *e.g.,* program understanding, development and maintenance [11, 16]. In contrast, our study provides a comprehensive view of the documentation issues encountered by practitioners.

Moreover, since our goal is to further research in the context of documentation recommender systems, the second contribution of this paper is an insight into the types of documentation that practitioners perceive as useful when confronted with specific software engineering tasks. Therefore, we formulate our second RQ as:

RQ_2: *What types of documentation are perceived as useful by practitioners in the context of specific software engineering tasks?*

To answer these two research questions, we performed two surveys with 146 professional software practitioners. In the first survey, we focused on the documentation issues that practitioners perceive as more relevant, together with the solutions they apply when these issues arise. In the second survey, we studied the types of documentation that practitioners consider important given specific tasks. Most participants (125) are from a multinational corporation active in automation technology, others (21) have been recruited in specialized online forums. The result is a diversified population of practitioners acting in various roles (*e.g.,* developers, testers).

The body of knowledge provided by the findings of these surveys will allow the research community to prioritize the practitioners' needs and to orient future efforts for the design and development of better automated documentation recommendation systems.

Table 1: Summary of previous studies about software documentation issues and relevant documentation types.

Study	Scope	Participants	Summary of findings
Forward and Lethbridge (2002) [14]	Documentation tools	48 individuals in the software field ranging from junior developers to managers and project leaders	Documentation tools should seek to better extract knowledge from core resources, including the system's source code, test code and changes to both. Resulting technologies could then help reduce the effort required for documentation maintenance.
Kajko-Mattsson (2005) [25]	Documentation and software maintenance	Developers of 18 Swedish organizations	Documentation is usually neglected during *corrective maintenance.*
De Souza *et al.* (2005) [11]	Documentation and software maintenance	76 software maintainers	Source code, code comments, data models, and requirement specification are perceived as the most important documents by maintainers.
Chen and Huang (2009) [8]	Documentation and software maintenance	137 project managers and software engineers	Lack of *traceability, untrustworthiness, and incompleteness* are among the most important documentation issues for maintainers.
Robillard (2009) [54]	Documentation and its effects on learning APIs	80 professionals at Microsoft	API learning resources are critically important when considering obstacles to learning the API. Information about the high-level design of the API is necessary to help developers choose among alternative ways to use the API, to structure their code accordingly, and to use the API as efficiently as possible. Code examples can become more of a hindrance than a benefit when there is a mismatch between the tacit purpose of the example and the goal of the example user.
Dagenais and Robillard (2010) [10]	Maintenance/evolution of documentation	22 developers or technical writers	Updating documentation with every change leads to a form of embarrassment-driven development, which in turn leads to an improvement in the code quality. Moreover, constant interaction with the projects' community positively impacted the documentation.
Robillard and Deline (2011) [55]	Documentation and its effects on learning APIs	440 professional developers at Microsoft	Some of the most severe obstacles faced by developers learning new APIs pertained to the documentation and other learning resources. Important factors to consider when designing API documentation: documentation of intent; code examples; matching APIs with scenarios; penetrability of the API; and format and presentation.
Garousi *et al.* (2013, 2015) [15, 16]	Documentation and software maintenance	25 professionals at NovAtel	Developers tend to use design documents during the development phase, while code comments are considered the most useful artifacts for maintenance. Readability, relevance of content, and organization are the quality attributes with the strongest impact on the overall perceived quality of documentation.
Plösch *et al.* (2014) [49]	Documentation quality	88 software development project members of various software organizations	*Accuracy, clarity, consistency, readability, structuredness,* and *understandability* are considered the most important documentation attributes. Documentation standards (*e.g.,* IEEE Std.1063-2001, ISO 26514:2008) are not perceived as important by developers.
Uddin and Robillard (2015) [69]	Common documentation problems	323 IBM software professionals	Cataloged and examined how ten common documentation problems manifested themselves in practice assessed those problems' frequency and severity. The most pressing problems were related to content, as opposed to presentation.
Sohan *et al.* (2017) [61]	Examples in API documentation	26 software engineers	REST API client developers face productivity problems with using correct data types, data formats, required HTTP headers and request body when documentation lacks usage examples.

2 RELATED WORK

The two major lines of research related to our work are devoted to (i) developing tools and approaches to automatically generate or recommend documentation, and (ii) empirically investigating different documentation aspects (*e.g.,* quality) and their impact.

Concerning the development of automated approaches, representative examples are the summarization techniques that provide abstractive and/or extractive summaries of software artifacts such as bug reports [36, 37, 52], classes and methods [19, 34, 38–40, 44, 58, 63], unit tests [29, 30, 48], code changes [9, 24, 32, 46, 47], user reviews [12], code snippets [72], and user stories [28]. Equally important are recommenders that support developers in finding APIs and code usage examples [21, 22, 31, 45, 64], code fragments implementing specific features [41, 51, 53, 66] or generally useful crowdsourced knowledge for a given implementation task [50].

Closer to our work is the empirical research on software documentation aspects, particularly the studies that interview or survey software practitioners. An overview of these studies is presented in Table 1. While some of them examine the importance and usage of documentation in specific phases of the software lifecycle [8, 11, 15, 16, 25], their focus is on general software engineering activities (*e.g.,* maintenance), as opposed to the specific tasks (*e.g.,* refactoring, debugging) that we investigate in our study. Moreover, the variety of tasks and documentation types that we consider is unmatched and provides empirical knowledge still needed to further research on context-aware recommendation systems [6, 56].

Also related is our previous work to use mining-based strategies to identify documentation issues discussed by practitioners [1], by developers [5, 33, 59], or by application users [26]. We [1] presented an extensive taxonomy of 162 types of issues faced by developers and users of software documentation.

We use our previous work as starting point to assess the relevance of the documentation issues presented in our taxonomy, which was neglected in our study. In other words, while our main target was the definition of a comprehensive taxonomy of documentation issues, in this work, our goal is to understand whether (and which of) those issues are relevant to practitioners. Compared to previous work investigating documentation issues from the practitioners' perspective [54, 55, 69], while it focused on a specific type of documentation (*i.e.,* API documentation), we consider a wide set of 51 documentation issues affecting various kinds of documentation.

Also worth mentioning is the mapping study by Zhi *et al.* [73] that reviews 69 documentation-related papers from 1971 to 2011. As a result, it was found that *completeness*, *consistency*, and *accessibility* are the most frequently discussed documentation quality attributes in the existing literature. Two conclusions are derived from that study: (i) some documentation aspects such as quality, benefits, and cost are often neglected, and (ii) more estimation models and methods are needed. Finally, Zhi *et al.* call for more and stronger empirical evidence, larger-scale empirical studies, and more industry-academia collaborations to investigate the software documentation field. Our study goes exactly in this direction.

3 STUDY DESIGN

The *goal* is to investigate the perception of practitioners of (i) the relevance of documentation issues, and (ii) the usefulness of different types of documentation in the context of specific tasks. The study *context* consists of *objects*, *i.e.,* two surveys designed to investigate the study goals, and *subjects* (referred to as "participants"), *i.e.,* 146 practitioners, 125 employed in a multinational corporation, and 21 recruited in specialized online forums.

3.1 Research Questions

We aim at answering the following research questions:

RQ_1: *What documentation issues are relevant to practitioners?* This research question builds on our taxonomy of documentation issues [1], which consists of 162 issues derived from the qualitative analysis of 878 documentation-related artifacts (*e.g.,* Stack Overflow discussions, pull requests). Although our taxonomy seems comprehensive, we did not investigate which documentation issues are actually relevant to practitioners. RQ_1 aims at filling this gap. Knowing the documentation issues that practitioners consider relevant can guide researchers in the development of techniques/tools aimed at identifying and possibly fixing these issues, rather than others not relevant to practitioners. It could also inform documentation writers and maintainers about quality attributes of documentation that must be prioritized.

RQ_2: *What types of documentation are perceived as useful by practitioners in the context of specific software engineering tasks?* This research question studies the types of software documentation (*e.g.,* code comment, release notes) that are considered useful by practitioners when performing a specific software-related activity (*e.g.,* code refactoring, debugging). This information is essential in the design of the next generation of software documentation recommender systems, whose goal is to automatically generate documentation customized for a given task [6, 56]. RQ_2 can shed some light on the type of information and documentation needed by practitioners under specific scenarios.

3.2 Context Selection: Surveys & Participants

Surveys. Figure 1 depicts the flow of our two surveys. The numbered white/gray boxes depict steps in which participants answer questions, and the black boxes represent either static information pages or an activity automatically performed by the survey application to select the next question to ask. The gray dashed box represents a loop of questions/answers performed repeatedly.

Both surveys have been implemented in Qualtrics (https://www.qualtrics.com) and start with a welcome page explaining the goal of the study, that the surveys are anonymous, and that they are designed to last *ca.* 15 minutes each. Once the participant agrees to start, both surveys show a form with basic demographic questions. In particular, we ask the participant's role in the software projects they contribute to (*e.g.,* developer, tester) and their years of experience in programming on a four-point scale: <3 years, 3-5 years, 5-10 years, >10 years. Once done with the collection of the demographic information, the two surveys differ (see Figure 1).

Survey-I. To design *Survey-I* (Figure 1-a), we revisited our taxonomy of 162 documentation issues (see Figure 1 in [1]) and identified the issues to be considered in our survey. This taxonomy is organized into four categories:

(1) **Information Content (What)** refers to problems with the documentation content (*i.e.,* "what" is written in the documentation). This is the predominant category in the taxonomy, with 55% of the analyzed documentation-related artifacts discussing these issues. It is organized into three subcategories: *Correctness* issues (*e.g.,* erroneous code examples), *Completeness* issues (*e.g.,* missing code behavior clarifications), and *Up-to-dateness* issues (*e.g.,* behavior described in the documentation is not implemented).

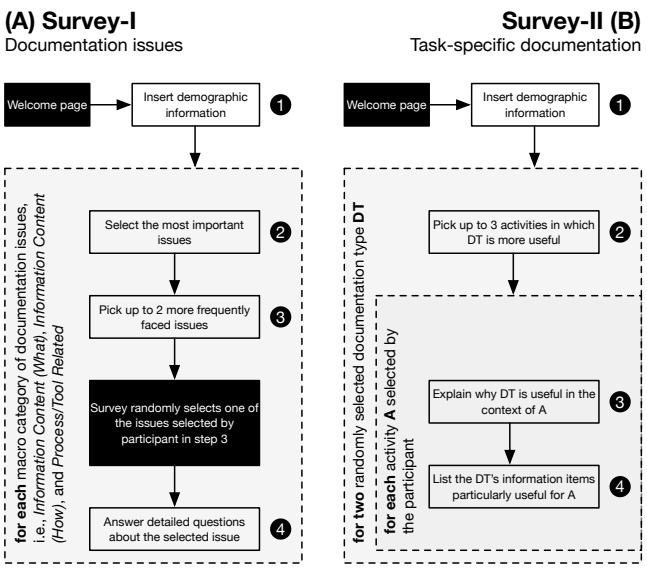


Figure 1: Design of the two surveys used in our study.

(2) **Information Content (How)** relates to "how" documentation is written and organized, and appears in 29% of the analyzed artifacts. Its subcategories capture issues with different documentation quality attributes, including *Maintainability* (*e.g.,* cloned documentation), *Readability* (*e.g.,* confusing documentation title), *Usability* (*e.g.,* poor support for navigating the documentation), and *Usefulness* issues (*e.g.,* code example needs improvement to become useful).

(3) **Process Related** groups issues linked to the documentation process, which were found in 9% of the analyzed artifacts. The issues in this category are organized into five subcategories, namely, *Internationalization* (*e.g.,* missing translation for a language), *Contributing to Documentation* (*e.g.,* unclear how to report issues found in the documentation), *Doc-generator Configuration* (*e.g.,* ignored Javadoc warnings), *Development issues caused by documentation* (*e.g.,* problems introduced by autogenerated comments), and missing *Traceability* information.

(4) **Tool Related** refers to issues associated with documentation tools, which were discussed in 15% of the analyzed artifacts. Subcategories include *Bugs* in documentation tools, lack of *Support*, unmet feature *Expectations*, usage difficulties (aka *Help required*), and *Migration* problems across different tools.

The taxonomy is hierarchically organized, meaning that each category (*e.g., Information Content (What)*) contains subcategories (*e.g., Correctness*) further organized into subcategories on many levels (*e.g., Correctness* includes five subcategories, one of which is further organized into two subcategories, leading to a total of four hierarchical levels).

Considering all the types of documentation issues composing our taxonomy [1] was not an option for our study. Indeed, asking a participant to read the complete list of 162 issues and pick the ones that are more relevant is excessive and would have led to a substantial increase in the survey abandonment rate.

For this reason, we limited the number of documentation issues considered in *Survey-I* by adopting the following process. First, we organized the survey into three parts: *Part I* focuses on issues related to *Information Content (What)*; *Part II* concentrates on *Information Content (How)* issues; and *Part III* investigates *Process-Related* and *Tool-Related* issues, together. Second, given the hierarchical organization of the taxonomy, we grouped together documentation issues that are very similar and share the same parent category, with the goal of reducing the overall number of issues to investigate. For example, in our taxonomy [1], issues related to inconsistency between code and documentation (*i.e.,* the code implements a behavior different from the one described in the documentation) are categorized into two subcategories, namely *behavior described in the documentation is not implemented*, and *code must change to match the documentation*. We decided to only consider the parent category, *Code-documentation inconsistency*. This grouping was done by two of the authors, and reviewed in multiple rounds by other three authors until agreement was reached.

The complete list of considered issues is available in our replication package [3]. Overall, we summarized the 162 issues from the original taxonomy into 51 documentation issues: 22 in the *Information Content (What)* category, 12 in the *Information Content (How)* category, and 17 in the combined *Process/Tool Related* category.

For each of the three parts of *Survey-I*, we show the list of related issues to the participant, asking them to select via checkboxes the ones they perceive as major issues (step ❷ in Figure 1-a). When hovering the mouse over the name of a specific issue, its brief definition pops up. We also provide an open field "Others", in which the participant could list documentation issues that were not listed in the predefined options. After that, we show the list of issues selected in the previous step as being important, but this time we ask them to select up to two issues that they face most frequently when reading/writing documentation (step ❸). Given this selection, the survey platform randomly picks one of the issues selected in step ❸ to collect detailed information about it. In particular, we ask in step ❹: (i) whether the specific issue concerns more the readers or the writers of the documentation, with possible choices on a five point scale (*i.e.,* only readers, mostly readers, equally both, mostly writers, only writers); (ii) how frequently the issue is encountered by the participant (possible choices: every day, 2-3 times per week, once a week, less often than once a week, never); (iii) what the solution is for the specific issue (open answer); and (iv) to describe the situation in details (optional, open answer). Note again that steps ❷ to ❹ were performed three times, one for each macro category of documentation issues.

Survey-II. Concerning the second survey (Figure 1-b), since our goal is to investigate the types of documentation useful in different software engineering tasks, we had to define the list of documentation types and tasks to consider. In the case of the documentation types, we started again from our taxonomy [1]. For each of its 162 documentation issue types, we annotated any type of documentation mentioned, *e.g.,* from the taxonomy node *Inappropriate installation instructions* we extracted the *installation guide* documentation type. The resulting list was then complemented and refined through face-to-face meetings among three of the authors. In particular, documentation types missing in the taxonomy but known to the authors were added, which led to the final list consisting of: *API Reference*, *Code Comment*, *Contribution Guideline*, *Deployment Guide*, *FAQ*, *How-to/Tutorial*, *Installation Guide*, *Introduction/Getting Started Document*, *Migration Guide*, *Release Note/Change Log*, *User Manual*, *Video Tutorial*, and *Community Knowledge*. For each documentation type, we provided a description and examples of information items usually contained in it. For example, the documentation type *Code Comment* is described as "Code Comments summarize a piece of code and/or explain the programmer's intent"; and the corresponding text for examples of information items is "Comments used to describe the functionality & behavior of a piece of code, the parameters of a function, the purpose and rationale for a piece of code". The descriptions and information items for all the documentation types can be found in our replication package [3].

Concerning the software engineering tasks, we started from the list of activities reported in the Software Engineering Body of Knowledge (SWEBOK) version 3.0 [7]. We went through the SWEBOK knowledge areas (*e.g.,* requirements, construction, maintenance) looking for activities that require, involve or produce documentation. The initial list was discussed by all authors to refine the terms for better comprehension of the participants when reading the survey. The final list of tasks used in *Survey-II* consists of: *Requirements Engineering*, *Software Structure and Architecture Design*, *User Interface Design*, *Database Design*, *Quality Attributes Analysis and Evaluation*, *Programming*, *Debugging*, *Refactoring*, *Program Comprehension*, *Reverse Engineering and Design Recovery*, *Software/-Data Migration*, *Release Management*, *Dealing with Legal Aspects*, *Software Testing*, and *Learning a New Technology/Framework*.

Once we defined the types of documentation and the tasks, we designed the survey. In *Survey-II*, after filling up their demographic information, the participant is shown with the name and description of a randomly selected documentation type *DT*. The survey asks the participant to select up to three software-related activities in which *DT* is considered more useful (step ❷ in Figure 1-b). For each selected activity *A*, two open questions are asked: (i) explain why *DT* is useful in the context of *A* (❸ in Figure 1-b); and (ii) list the information items in *DT* that are particularly useful for *A* (❹ in Figure 1-b). Step ❷ and the loop including steps ❸ and ❹ are performed twice for two randomly selected documentation types.

We tested both surveys with four developers and four PhD students to check that the questions were clear and that each survey could be completed within 15 minutes, a duration agreed upon with the partner company. As a consequence of this pilot study, we rephrased a number of questions and set the "thresholds" used in our surveys (*e.g.,* only ask detailed questions about one of the issues selected by participant in step ❸ of *Survey-I*).

Participants. We started by collecting answers from the practitioners of the partner company. We invited participants via an email that summarized the goal of the study and contained the link to our surveys. There was a single link to both surveys, but the application automatically assigned a participant to one of them, balancing the number of data points per survey.

As a first test batch, we invited 160 practitioners collecting 21 responses. As no problems were detected, we emailed the invitation to 1,500 practitioners and posted an announcement on the Yammer service of the company. We received 104 additional answers, leading to a total of 125 completed surveys (incomplete surveys were discarded), 65 for *Survey-I* and 60 for *Survey-II*.

Computing a response rate for our study is difficult due to the posted announcement, and to the fact that the survey was conducted over the summer. Assuming that all the 1,660 practitioners received and opened our email, and ignoring that other practitioners were possibly reached through the Yammer service, this would result in a 9.5% response rate, which is in line with the suggested minimum response rate of 10% for survey studies[17]. We collected a slightly higher number of responses for *Survey-I* as compared to *Survey-II* (*i.e.,* 65 *vs* 60). This is due to the fact that some participants started the study simultaneously (thus being equally distributed across the two different surveys by the platform), but some of them did not finish the assigned survey, thus unbalancing the final numbers.

We also posted the link to our survey on social websites oriented to developers and programming. This allowed us to collect 21 additional complete answers, leading to the final 146 answers to our surveys, 78 to *Survey-I* and 68 to *Survey-II*. An overview of the surveyed participants and their experience is depicted in Table 2.

Table 2: Participants roles & programming experience

Role	Population	<3y	3-5y	5-10y	>10y
Developer	55	12	8	10	25
Architect/Technical Engineer	26	1	1	2	22
Technical Lead	19	0	0	6	13
Test Analyst/Tester/Test Engineer	11	1	0	7	3
Others	35	4	2	4	25
	146	18	11	29	88

3.3 Data: Analysis & Replication Package

Analysis. We answer RQ_1 by relying on descriptive statistics. For each of the predefined documentation issues and for those added through the "Other" field, we report the percentage of participants that perceives it as an important concern. We also report on how frequently participants face the issues considered as important, and whether they affect more documentation readers or writers. Moreover, we qualitatively discuss interesting cases shared by participants about real instances of these issues and the solutions they adopted to address them. To analyze the participants' solutions (❹ in Figure 1-a), we followed an open-coding inspired approach, where two of the authors independently assigned a tag to each of the 101 answers that described solutions to documentation issues. The tag was meant to summarize the described solution (*e.g., improve project management practices* derived from *"Specifically allocate efforts for documentation in the task planning"*). Conflicts were solved through a face-to-face meeting.

To answer RQ_2, we analyze a heat map (Figure 3) depicting, for each activity type (rows), the percentage of participants that indicated each documentation type (columns) as useful in that context. Then, we qualitatively discuss the reasons provided by participants (❸ in Figure 1-b) to explain why a documentation type is useful during a specific activity. For each documentation type, we also report the information items listed by participants as particularly useful in each of the software engineering tasks that we investigated.

Replication Package. All material and data used to run our study as well as the developers' anonymized answers are available in our replication package [3].

4 WHAT DOCUMENTATION ISSUES ARE RELEVANT TO PRACTITIONERS?

Figure 2 summarizes the responses collected for *Survey-I*. We discuss the practitioners' perspective about the documentation issues listed in the three main categories (*i.e.,* "Information Content (What)", "Information Content (How)", and "Process/Tool Related"). We highlight lessons learned and recommendations for researchers (💡), and confirm/refute previous findings reported in the literature (🔗).

Information Content (What). We observe in Figure 2 that all issues in this category are perceived as important by practitioners.

🔗 This result is in line with previous studies [8, 69] that underlined the relevance of correctness, completeness, and up-to-dateness issues in documentation.

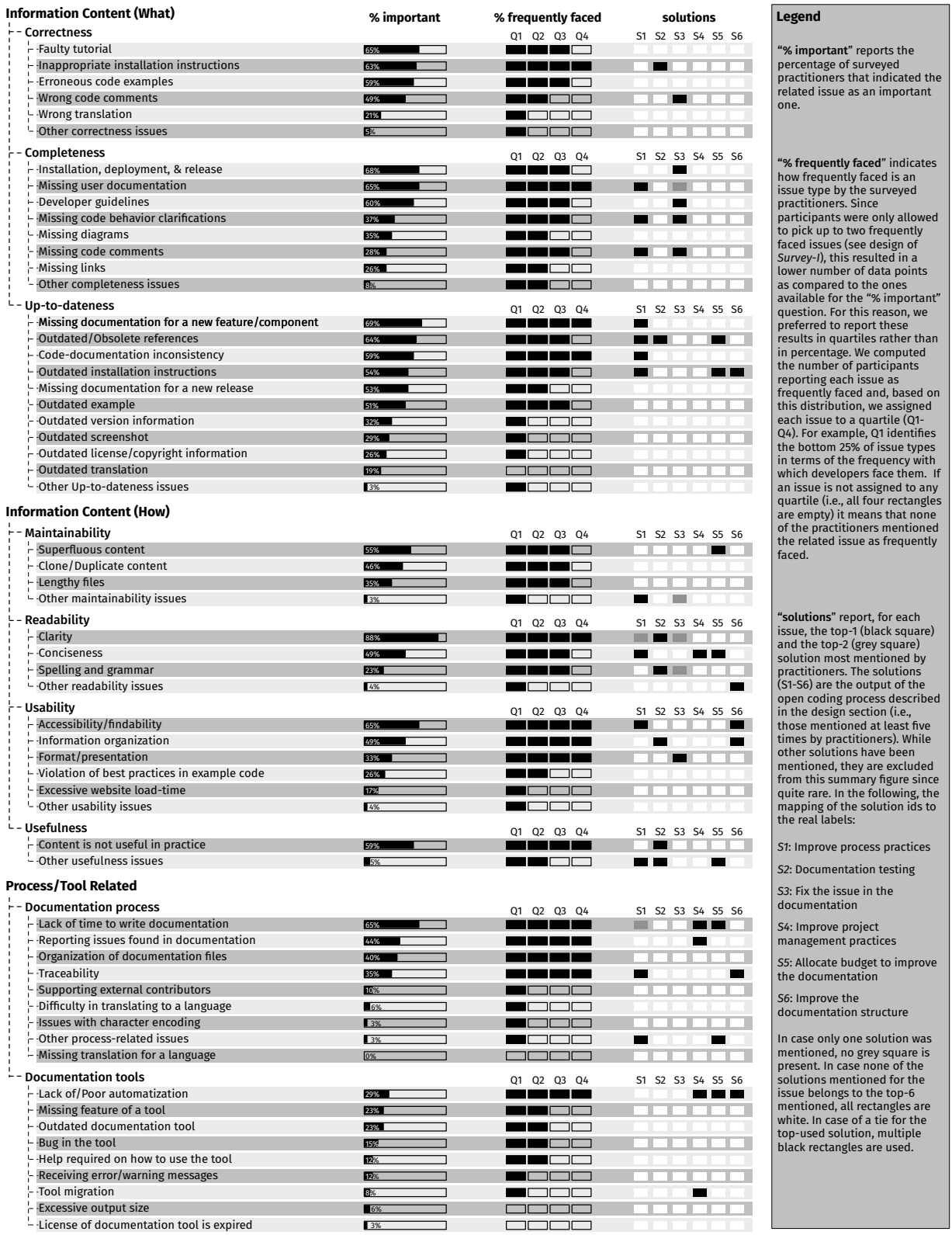
Regarding the *Correctness* subcategory, all its issues except *wrong translation* were considered to be important by at least half of the surveyed participants. Among them, *inappropriate installation instructions* was the most frequently encountered issue and, together with *faulty tutorial*, the one considered relevant by most practitioners (65% of them). Participants suggested a few possible solutions for this issue, such as performing reviews on the installation instructions by both internal team members and external users, who can provide feedback about the quality and usefulness of the document. Lightweight virtualization approaches (*e.g.,* Docker containers) can support practitioners in the creation of diverse reusable deployment environments to test installation instructions.

> 💡 The fragmentation problem of running environments for software is well-known (see the case of the Android ecosystem [20, 35]), which might lead to unexpected race conditions or compilation issues under specific platforms [67]. Research efforts could be devoted to (automated) testing of installations instructions under different environments, or automated generation of installation instructions. Our survey shows that this is an area of interest to practitioners, and even limited support for testing installation instructions across diverse environments would be welcome.

Wrong code comments is perceived as important by almost half (*i.e.,* 49%) of the surveyed practitioners. Besides the obvious (fixing the comment), practitioners also suggested to train the comments' writers, particularly in their technical English language skills.

> 💡 Code comments can be incorrect due to inaccurate information, as well as to the writer's inability to clearly describe a code fragment or change rationale in English. This is confirmed by the answers to the *wrong translation* issue. Some participants attributed it to documentation sometimes written by non-native English speakers. A suggested solution is to host the documentation on collaborative platforms (*e.g.,* Wikis) or code-sharing platforms (*e.g.,* GitHub), encouraging and enabling external contributors to add new content or (recommend how to) fix errors in the documentation.

Erroneous code examples were also recognized as an important issue by the surveyed participants (59% of them). It is well-known that code examples are a main information source for developers [55]. Facilitating error reporting, *e.g.,* by adding a comment section below each documentation page, was a suggested solution.



Legend

"% important" reports the percentage of surveyed practitioners that indicated the related issue as an important one.

"% frequently faced" indicates how frequently faced is an issue type by the surveyed practitioners. Since participants were only allowed to pick up to two frequently faced issues (see design of *Survey-I*), this resulted in a lower number of data points as compared to the ones available for the "% important" question. For this reason, we preferred to report these results in quartiles rather than in percentage. We computed the number of participants reporting each issue as frequently faced and, based on this distribution, we assigned each issue to a quartile (Q1-Q4). For example, Q1 identifies the bottom 25% of issue types in terms of the frequency with which developers face them. If an issue is not assigned to any quartile (i.e., all four rectangles are empty) it means that none of the practitioners mentioned the related issue as frequently faced.

"solutions" report, for each issue, the top-1 (black square) and the top-2 (grey square) solution most mentioned by practitioners. The solutions (S1-S6) are the output of the open coding process described in the design section (i.e., those mentioned at least five times by practitioners). While other solutions have been mentioned, they are excluded from this summary figure since quite rare. In the following, the mapping of the solution ids to the real labels:

S1: Improve process practices

S2: Documentation testing

S3: Fix the issue in the documentation

S4: Improve project management practices

S5: Allocate budget to improve the documentation

S6: Improve the documentation structure

In case only one solution was mentioned, no grey square is present. In case none of the solutions mentioned for the issue belongs to the top-6 mentioned, all rectangles are white. In case of a tie for the top-used solution, multiple black rectangles are used.

Figure 2: Documentation issues that are relevant to practitioners (RQ_1), according to the results of Survey-I.

We [1] suggested the development of testing techniques tailored to code examples. Testing activities on code examples, however, were not mentioned by the surveyed developers.

Among the issues related to *Completeness*, the lack of *installation, deployment, & release* instructions, *user documentation* (*e.g.,* user manual), and *developer guidelines* were considered important by a majority (respectively 68%, 65% and 60%), and are frequently encountered issues. Practitioners highlighted the importance of considering the creation of these types of documentation as first-class citizens, and suggested to include these documentation types as mandatory items in the release checklist and allocate project budget and a dedicated team to fundamental documentation items.

> Increasing the budget dedicated to documentation was a recurring solution often mentioned by participants. This suggests that software documentation does not receive the attention it deserves when planning and allocating software resources. This finding is relevant to software effort estimation models [43] that should explicitly consider the cost of documentation as one of the factors impacting the final effort needed to build a software system.

The low number of participants (28% of them) who indicated *missing code comments* as a major concern was unexpected. Previous studies [11] confined the importance of code comments to general software engineering activities. Some practitioners attributed missing comments to understaffed projects, where the team tends to focus more on coding rather than on documenting. However, they highlighted the need for *"instilling discipline into the team and encouraging writing code comments as a good coding practice"*. Practitioners also indicated the need for automation in code comment generation, *e.g.,* generating templates for bootstrapping the writing process and/or automatically generating (part of) code comments. The latter is an active research area [29, 30, 34, 38–40] and a roadmap has been proposed by Robillard *et al.* [56].

The results of our survey confirm the potential and need of automated documentation generation techniques. An exception was an answer provided by a practitioner advocating for writing code in a *"self-explanatory way, with as few comments as possible"*.

Automated tools are also invoked by practitioners to address issues caused by *missing diagrams*. One of them suggested that *"it should be easier to create graphs/diagrams from the text"*.

> While approaches for generating diagrams from low-level artifacts (*e.g.,* source code) exist (see, *e.g.,* [27]), practitioners call for approaches that support the extraction of diagrams (*e.g.,* components diagrams) from high-level text-based artifacts (*e.g.,* requirements). The development of these techniques poses interesting research challenges, such as the identification of components needed to implement a given requirement as well as their interactions. This represents an interesting direction for future research.

Regarding *Up-to-dateness* issues, the *lack of documentation for a new feature/component* was not only the one considered important by most participants (69% of them) but also the most recurring issue in this subcategory. Practitioners tend to resort to external sources of information to understand the new feature, or to contact the appropriate parties (*i.e.,* the team who developed the feature) to ask for explanations. One participant stressed the importance of documenting code implementing new features by using concrete examples: *"some parameters are impossible to understand without documentation, and documentation is often not very good"*. Here, automated documentation tools appear again as a solution to enable automated refactoring of documentation, and up-to-date documentation generation as part of continuous integration pipelines.

Inconsistency between code and documentation was also perceived as an important issue by practitioners (59% of them) and is one of the top recurring issues they face.

This observation is in line with previous studies [8, 49, 69], which found documentation consistency to be a major issue. The most common up-to-dateness documentation issues involve code comments, which might not reflect changes implemented in the code [71]. An interesting observation is related to the maintainability of code comments: One practitioner highlighted that a solution for up-to-dateness issues is to limit code comments to the minimum needed, so it is simpler to co-evolve them with code (*i.e.,* if a comment documents useless details, it is more likely that changes implemented in the code will impact it).

> The research community has focused on the generation of code comments, while our survey points to the need for approaches that identify redundant and/or unnecessary code comments that increase the comment maintenance cost. Maintainability of comments is a real concern.

While other *Up-to-dateness* issues were considered important by practitioners (see Figure 2), exceptions to this trend were: *outdated license/copyright information*, *outdated screenshot*, *outdated translation*, and *outdated version information*.

Summing up. In the *Information Content (What)* category, 7 out of 23 (30%) documentation issues from our taxonomy [1] are perceived as important by the majority (≥60%) of surveyed practitioners (*e.g., faulty tutorial*, *inappropriate installation instructions*, *missing documentation for a new feature/component*). Instead, nine issues (39%) are considered relevant by less than 40% practitioners—see *e.g., wrong translation*, *outdated screenshot*, *outdated license*. This is a first indication that, despite the comprehensiveness of our taxonomy [1], the research community could prioritize selected issues that are actually relevant to practitioners.

Information Content (How). This category of issues is related to the way documentation content is written and organized. Regarding *Maintainability* issues, practitioners considered *superfluous content* (55% of them) and *clone/duplicate content* (46%) the main sources of concern. This observation is in line with our previous study [1], which reports that these two subcategories are responsible for ~71% of developers' discussions on maintainability of documentation.

> Given the frequency of these issues [1] and their relevance to practitioners, the research community could leverage existing technologies to provide (even partial) solutions. For example, as code clone detection approaches have been defined in the literature [60], similar techniques using natural language processing could be developed to identify (and suggest how to refactor) cloned content in software documentation.

> In the case of *superfluous content*, a compelling next step would be to qualitatively study this type of content to develop techniques for its automatic detection (similarly to the work in code smell detection, with a combination of empirical studies [68] and detection techniques [42]).

Concerning *Readability*, documentation *clarity* is the issue perceived as most important by practitioners (88% of them), as it affects them in many ways (*e.g., "A developer in our team created confusing and overly complicated documentation for customers of our solution"*, *"We experienced this issue when deploying an app that was built by a third party that no longer supports us; the documentation they provided is not clear on how to configure it properly"*).

Previous studies also reported the importance of *Readability* [15, 16, 49, 49] and *Understandability* [49].

To deal with this issue, a number of solutions were proposed by participants. First, documentation writers should always keep in mind the actual documentation users and their needs when writing a document. Second, documentation should be tested by someone with little domain knowledge. For example, if the documentation at hand is an installation guide, it should be tested by users of the system rather than a development team member. This, according to the practitioners, would help in promoting documentation clarity. Moreover, one participant stressed the importance of selecting essential information items in the documentation, highlighting them, and investing in their writing. Our second research question investigates the information items, from different types of documentation, that are more useful during specific software engineering activities.

In the *Usability* subcategory, issues related to *accessibility/findability* and *information organization* are considered important by practitioners (65% and 49% of them, respectively), while others (*e.g., excessive website load-time*, *violation of best practices in example code*) are not perceived as such.

These findings are in line with our previous results [1]. Other previous works [15, 16, 49, 69] have revealed the importance of documentation organization and its impact on content findability, confirming the relevance of these issues.

> Studies on how users interact with and search documentation could help the research community to define clear guidelines on how different types of documentation should be organized to address accessibility/findability issues.

Format/presentation issues in the documentation are frequently encountered by practitioners who, however, do not consider them as a major issue (only 33% of the participants perceived them as important). The most common suggested solution for this type of issues is to provide documentation guidelines and standard templates. Tools to validate documentation format and adherence to a predefined template would be useful in this context.

In the *Usefulness* subcategory, 59% of the surveyed practitioners considered *content is not useful in practice* as a major and frequent issue. Reviewing the documentation before release, and providing more in-depth details and practical examples were suggested solutions to deal with this issue. The prevalence of this usefulness issue stresses further more the importance of our second research question, *i.e.,* knowing the information items that are actually useful for practitioners can help in avoiding useless content.

Summing up. In the *Information Content (How)* category, only 2 out of 12 (17%) documentation issues from our taxonomy [1] are perceived as important by at least 60% of surveyed practitioners (*i.e., clarity* and *accessibility/findability*).

Process/Tool Related. Compared to the previous two categories, developers found issues related to the documentation process and tools less important (see Figure 2). One substantial difference between this category and the previous two is that, as indicated by participants, issues in this category mostly affect documentation writers, while both writers and readers are affected by most of the issues in the other categories.

Lack of time to write documentation is the only issue in this category that was indicated as important by the majority (65%) of participants. Also, it is a frequently encountered issue. The proposed solutions boil down to: (i) explicitly allocating time/effort/resources to documentation in the project planning; and (ii) starting documentation activities in the early stage of the software lifecycle, to avoid situations such as the one described by a practitioner: *"The documentation team comes into picture only at the last moment before the release"*. The consequences of inadequate documentation planning were also stressed by another practitioner: *"Projects were built without providing documentation; as time passes aspects of the project are forgotten which makes revisiting the project when updates or modifications need to be made more difficult"*. As highlighted by a respondent, *"when estimating time to complete a project, it is important to make sure that documentation is counted in"*.

Among other *Process/Tool Related* issues, poor *organization of documentation files* and *traceability* issues were frequently encountered by developers, even though only 40% and 35% of them, respectively, considered these issues important.

> Lack of traceability in documentation has also been reported in previous work [8]. This issue could be addressed by investing in professional tools that integrate traceability recovery techniques proposed in the literature [4].

Regarding *Documentation tools* issues, the *lack of/poor automatization* was the only issue frequently faced by practitioners. Smarter tools, better IDE integration, and automated generation of documentation were the common requests in this context.

> The need for automation can be justified by the *lack of time* issue discussed above. The research community is already investigating novel techniques for the automatic generation of documentation [56], and our survey confirms the practical relevance of this research area to practitioners.

Participants mentioned other tool-related issues, such as the lack of training for teams or the lack of good tool support for some languages: *"Writing good docs for C/C++ is hard; there are no tools that capture function/class semantics [...]; this would allow the automation of at least a part of the doc writing process"*.

Summing up. Concerning the *Process/Tool Related* category, the majority of issues in our taxonomy [1] are not considered important. The notable exception is represented by the *lack of time to write documentation.* Four additional issues are frequently faced by practitioners, including the *lack of/poor automatization.*

5 WHAT TYPES OF DOCUMENTATION ARE USEFUL TO PRACTITIONERS?

Figure 3 presents in a heat map the percentage of practitioners indicating documentation types (columns) as useful for given software engineering tasks (rows). A dark spot represents an artifact that was found to be particularly useful for a specific task (*e.g.,* *Code Comment* for *Software Debugging*), while a white spot shows that none of the participants considered the documentation artifact useful for a given task (*e.g., Code Comment* for *Database Design*).

We grouped and sorted the software engineering tasks based on the stage of the software lifecycle to which they relate the most. This resulted in three groups of tasks: *Requirements & Design*, *Development & Testing*, and *Operation & Maintenance*. We present a heat map that combines the data about the different tasks in the mentioned groups at the bottom of Figure 3.

The documentation types are sorted based on the average percentage of participants who considered the documentation type useful for each of the 15 tasks.

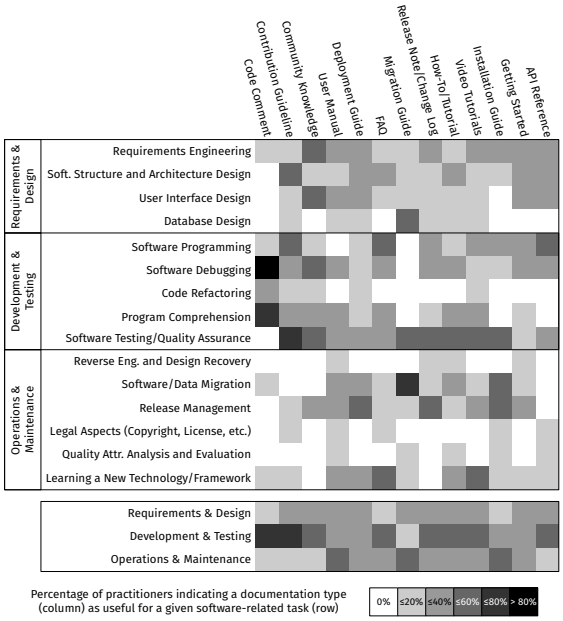


Figure 3: Types of documentation perceived as useful by practitioners in the context of specific software engineering tasks (RQ_2), according to the results of Survey-II.

In the following, we discuss interesting cases, while more detailed results are available in the replication package [3].

Code Comment and *Contribution Guideline* were the two documentation types considered as more useful for the different tasks. The distributions of answers are quite different and skewed towards different tasks. *Code Comment* was found highly useful for only a few *Development & Testing* tasks. In particular, all participants agreed on the usefulness of *Code Comment*s for *Software Debugging* and 80% of them also pointed to their importance for *Program Comprehension*. 40% of practitioners marked *Code Refactoring* as an activity benefiting from code comments. Concerning other tasks, excluding isolated exceptions, participants did not mark code comments as useful.

When asked about why *Code Comment*s are useful for the aforementioned tasks, participants emphasized that comments communicate information that is not evident in the code but could help other developers, particularly those who join the project at later stages: *"Software is developed over a long period of time by many different developers; something that may seem obvious to one person may not be obvious to the person who has to maintain the code"*. Concerning the useful information items from comments, participants who marked them as useful for debugging highlighted the fundamental role of parameters' descriptions, *e.g., "Comments might tell the meaning of parameter if the variable name is not adequate"*. Assumptions made in the code should be also documented since they serve in debugging. Practitioners mostly see comments as a way to gather information about the code purpose and behavior.

The different information items in code comments deemed useful for two quite related tasks (*i.e.,* code debugging and program comprehension) confirms that the context is essential in documentation recommender systems aimed at automatically generate documentation (*e.g.,* comments) or at pointing to useful sources of information [6, 56]. More in general, it highlights the importance of keeping comments updated and consistent with the source code, as also observed in our previous study [1].

> Only a few approaches are available to detect code-comment inconsistencies, but they are specialized to specific types of comments (*e.g.,* Javadoc [65]).

Contribution Guideline was found helpful in development tasks (64% of the participants found it practical for *Software Testing/Quality Assurance*, and 45% for *Software Programming*), but it is more versatile than *Code Comment* and suitable for design tasks too: 45% of participants claimed its usefulness during *Software Structure and Architecture Design*. They mentioned various benefits derived from the usage of this documentation, *e.g.,* consistent style and use of common programming techniques, improved productivity of new contributors, explanation of workflows, CI pipelines, and releases processes. Interestingly, a sales manager wrote about the importance of this documentation as a means to demonstrate to the customer the quality of the developed products: *"Specifically I deal with pre-sales and eventually end user. This document would validate our concerns and investments in the product [...], so it is very important"*. Looking at the useful information items from this document, practitioners mostly mentioned *best practices*, *coding style guidelines*, *testing requirements*, and *pull request/release checklist*. The last two were considered as particularly important for *Software Testing/Quality Assurance*.

> *Contribution Guideline* is an often neglected document that does not seem to have a negative effect when it is missing but, as shown by our survey, can positively influence a wide range of tasks when it is well written. It is also poorly considered in the researcher community; hence, there are many open possibilities to help practitioners. An interesting direction could be to (partially) generate such a document by automatically recognizing development practices, *e.g.,* coding styles, testing practices, frequently reviewed aspects in pull requests. Such a tool could benefit from learning approaches (*e.g.,* Allamanis *et al.* [2]).

Another documentation type useful for many tasks is the *User Manual*. It was found helpful for most of the tasks by at least one fifth of the participants. The most important information items from this document are, according to the surveyed participants: *Screenshots*, *Description of expected behavior*, and *Step-by-step technical descriptions*. However, also in this case, we observed differences in the distribution of these items depending on the task. For example, *Screenshots* and *Description of expected behavior* are useful for *Software Debugging* (to check how the system should behave).

🔗 Despite the apparent relevance of a *User Manual* in many tasks, our taxonomy [1] has only one issue (*Missing User Manual*) directly mentioning it, although other more general issues in the taxonomy could apply to this type of document. Given the recognized importance of *User Manual*s, researchers have started working on approaches to automatically generate/update parts of this document, *e.g.,* screenshots. Representative examples are the work by Waits and Yankel [70] and Souza and Oliveira [62].

There are several white spots in Figure 3, partially due to the low number of responses we collected for *Code Refactoring*, *Reverse Engineering and Design Recovery*, *Legal Aspects*, and *Quality Attributes Analysis and Evaluation*. Recall that participants could choose up to three tasks for each assigned documentation type, so it is possible that they did not select other tasks for which the given documentation might still be useful, but not as much as for the selected top-3. On average, participants selected 2.6 tasks per documentation item, so they indeed took the opportunity to select three tasks in some cases. This could explain the limited number of feedback for some tasks. It also explains why *API Reference* was found helpful in fewer cases than one might expect. Instead of selecting *Code Refactoring* or *Code Comprehension* development tasks, participants put their priorities on other tasks for this type of documentation.

🔗 For *API Reference*, most participants mentioned *Overviews/-Summaries of fields/methods* and *Code Examples* as relevant, in line with our previous findings [1], where we also found many issues related to code examples, and stressed the importance of ensuring the consistency of code examples and the actual code. It also supports the importance of research fields such as code summarization [18, 19, 39, 40] and the automatic generation of code examples [45].

Finally, the perceived usefulness of *How-To/Tutorials* for different tasks is noteworthy. This is expected when considering the increasing availability of online tutorials about many different topics.

Summing up. The main message resulting from RQ_2 is that practitioners perceive different documentation types as useful for different tasks. Also, as shown in some of the discussed examples, even within the same documentation type, different information items might be useful for different tasks. This supports the need for context-aware documentation recommender systems [6, 56].

6 THREATS TO VALIDITY

Construct validity. They are primarily related to the process we used to select the types of documentation issue (*Survey-I*) and the list of documentation types and tasks (*Survey-II*) for our surveys. These selections may not be representative of all possible documentation issues/types and software-related tasks. To mitigate this threat we always included a free-form "Other" option in the set of answers where these lists were used.

Internal validity. One factor is the response rate: while it does not look very high (9.5%), it is in line with the suggested minimum response rate for survey studies, *i.e.,* 10% [17].

Another possible threat concerns the fact that 146 respondents decided to participate to the survey because they had greater interest in documentation than others, thus providing a "biased view" of the investigated phenomena. However, our population is composed of practitioners having different roles and, as shown by the results, quite different views on the documentation issues and on the usefulness of different types of documentation for specific tasks.

Finally, a typical co-factor in survey studies is the respondent fatigue bias. We mitigated this threat by running a pilot study with four professional developers and four PhD students, to make sure that both surveys could be answered within 15 minutes.

External validity. The obvious threats here are (i) the context of our study, limited to participants mostly from a single multinational company, and (ii) the total number of participants (*i.e.,* 146). Concerning the first point, while developers from different companies/domains could have different views of the studied phenomena, we collected answers from 20 different countries across 4 continents (complete data in our replication package [3]). As for the number of participants, it is higher or in line with many previously published survey studies [10, 11, 14, 15, 25, 49, 54, 61].

7 CONCLUSION

We presented two surveys conducted with a total of 146 practitioners and aimed at investigating: documentation issues they perceive as important, and possible solutions they adopt when facing these issues (*Survey-I*); and documentation types they perceive as useful during specific software engineering tasks (*Survey-II*).

For *Survey-I*, we started from our previous taxonomy [1]. Said taxonomy had not been validated with practitioners, and indeed our first study showed that only a small subset of the 162 documentation issues reported in our taxonomy are deemed important by practitioners. Based on the survey responses, we provide a set of suggestions (💡) for future research endeavours, some of which are surprisingly low hanging fruits. In essence, it does not take much to ameliorate the state of affairs around documentation, and we believe that our now validated taxonomy represents a good starting point.

As for *Survey-II*, our results show when practitioners deem certain documentation types more important for specific tasks at hand. As the research community is headed towards the development of automated documentation generation recommenders [6, 56], we believe that our second study provides precious knowledge for the road ahead.

ACKNOWLEDGMENTS

Lanza and Aghajani gratefully acknowledge the financial support of the Swiss National Science Foundation for the projects PROBE (SNF Project No. 172799) and CCQR (SNF Project No. 175513). Aghajani also thanks CHOOSE (https://choose.swissinformatics.org) for the kind sponsorship.

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 *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. IEEE Press, 1199–1210.

[2] Miltiadis Allamanis, Earl T. Barr, Christian Bird, and Charles Sutton. 2014. Learning Natural Coding Conventions. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*. ACM, 281–293. https://doi.org/10.1145/2635868.2635883

[3] Anonymous. 2019. Replication Package. https://github.com/icse2020anonymousauthors/ReplicationPackage

[4] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. 2002. Recovering traceability links between code and documentation. *IEEE Trans. on Soft. Eng.* 28, 10 (2002), 970–983.

[5] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. 2014. What Are Developers Talking About? An Analysis of Topics and Trends in Stack Overflow. *Empirical Softw. Eng.* 19, 3 (2014), 619–654. https://doi.org/10.1007/s10664-012-9231-y

[6] Gabriele Bavota. 2016. Mining Unstructured Data in Software Repositories: Current and Future Trends. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 5. IEEE, 1–12.

[7] Pierre Bourque and Richard E. Fairley. 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0* (3rd ed.). IEEE Computer Society Press.

[8] Jie Cherng Chen and Sun Jen Huang. 2009. An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software* 82, 6 (2009), 981–992. https://doi.org/10.1016/j.jss.2008.12.036

[9] Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. 2014. On Automatically Generating Commit Messages via Summarization of Source Code Changes. In *Proc. of the IEEE 14th Int. Working Conf. on Source Code Analysis and Manipulation*. IEEE, 275–284. https://doi.org/10.1109/SCAM.2014.14

[10] Barthélémy Dagenais and Martin P. Robillard. 2010. Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors. In *Proc. of the 18th ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (FSE 2010)*. ACM, 127–136. https://doi.org/10.1145/1882291.1882312

[11] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. 2005. A Study of the Documentation Essential to Software Maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information (SIGDOC '05)*. ACM, 68–75. https://doi.org/10.1145/1085313.1085331

[12] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What Would Users Change in My App? Summarizing App Reviews for Recommending Software Changes. In *Proc. of the 24th ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (FSE 2016)*. ACM, 499–510. https://doi.org/10.1145/2950290.2950299

[13] Association for Computing Machinery (ACM). 2012. The 2012 ACM Computing Classification System. https://www.acm.org/publications/class-2012

[14] Andrew Forward and Timothy C. Lethbridge. 2002. The Relevance of Software Documentation, Tools and Technologies: A Survey. In *Proc. of the 2002 ACM Symp. on Doc. Eng. (DocEng)*. ACM, 26–33. https://doi.org/10.1145/585058.585065

[15] Golara Garousi, Vahid Garousi, Mahmoud Moussavi, Guenther Ruhe, and Brian Smith. 2013. Evaluating usage and quality of technical software documentation: an empirical study. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 24–35.

[16] Golara Garousi, Vahid Garousi-Yusifoglu, Guenther Ruhe, Junji Zhi, Mahmoud Moussavi, and Brian Smith. 2015. Usage and usefulness of technical software documentation: An industrial case study. *Information and Software Technology* 57, 1 (2015), 664–682. https://doi.org/10.1016/j.infsof.2014.08.003

[17] Robert M. Groves, Floyd J. Fowler Jr., Mick P. Couper, James M. Lepkowski, Eleanor Singer, and Roger Tourangeau. 2009. *Survey Methodology, 2nd edition*. Wiley.

[18] Sonia Haiduc, Jairo Aponte, and Andrian Marcus. 2010. Supporting Program Comprehension with Source Code Summarization. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2 (ICSE'10)*. ACM, 223–226. https://doi.org/10.1145/1810295.1810335

[19] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. 2010. On the Use of Automated Text Summarization Techniques for Summarizing Source Code. In *Proceedings of the 17th Working Conf. on Rev. Eng.* IEEE Comp. Soc., 35–44. https://doi.org/10.1109/WCRE.2010.13

[20] Dan Han, Chenlei Zhang, Xiaochao Fan, Abram Hindle, Kenny Wong, and Eleni Stroulia. 2012. Understanding Android Fragmentation with Topic Analysis of Vendor-Specific Bugs. In *2012 19th Working Conference on Reverse Engineering*. IEEE Comp. Soc., 83–92.

[21] Reid Holmes and Gail C. Murphy. 2005. Using Structural Context to Recommend Source Code Examples. In *Proc. of the 27th Int. Conf. on Software Engineering (ICSE 2005)*. ACM, 117–125. https://doi.org/10.1145/1062455.1062491

[22] Reid Holmes, Robert J. Walker, and Gail C. Murphy. 2006. Approximate Structural Context Matching: An Approach to Recommend Relevant Examples. *IEEE Transactions on Software Engineering* 32 (12 2006), 952–970. https://doi.org/10.1109/TSE.2006.117

[23] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep Code Comment Generation. In *Proceedings of the 26th Conference on Program Comprehension* (Gothenburg, Sweden). ACM, 200–210. https://doi.org/10.1145/3196321.3196334

[24] Siyuan Jiang and Collin McMillan. 2017. Towards Automatic Generation of Short Summaries of Commits. In *Proc. of the 25th IEEE/ACM Int. Conf. on Program Comprehension (ICPC 2017)*. IEEE Press, 320–323. https://doi.org/10.1109/ICPC.2017.12

[25] Mira Kajko-Mattsson. 2005. A Survey of Documentation Practice within Corrective Maintenance. *Empirical Software Engineering* 10, 1 (2005), 31–55. https://doi.org/10.1023/B:LIDA.0000048322.42751.ca

[26] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E. Hassan. 2015. What Do Mobile App Users Complain About? *IEEE Software* 32, 3 (2015), 70–77. https://doi.org/10.1109/MS.2014.50

[27] Elena Korshunova, Marija Petkovic, MGJ Van Den Brand, and Mohammad Reza Mousavi. 2006. CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code. In *2006 13th Working Conference on Reverse Engineering*. IEEE Comp. Soc., 297–298.

[28] Rrezarta Krasniqi, Siyuan Jiang, and Collin McMillan. 2017. TraceLab Components for Generating Extractive Summaries of User Stories. In *2017 IEEE Int. Conf. on Soft. Maint. and Evolution (ICSME)*. IEEE, 658–658. https://doi.org/10.1109/ICSME.2017.86

[29] Boyang Li, Christopher Vendome, Mario Linares-Vásquez, and Denys Poshyvanyk. 2018. Aiding Comprehension of Unit Test Cases and Test Suites with Stereotype-based Tagging. In *Proceedings of the 26th Conference on Program Comprehension (ICPC'18)*. ACM, 52–63. https://doi.org/10.1145/3196321.3196339

[30] Boyang Li, Christopher Vendome, Mario Linares-Vásquez, Denys Poshyvanyk, and Nicholas A. Kraft. 2016. Automatically Documenting Unit Test Cases. In *Proc. of the IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2016)*. IEEE, 341–352. https://doi.org/10.1109/ICST.2016.30

[31] Jing Li, Aixin Sun, and Zhenchang Xing. 2018. Learning to answer programming questions with software documentation through social context embedding. *Information Sciences* 448 (2018), 36–52.

[32] Mario Linares-Vásquez, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. ChangeScribe: A Tool for Automatically Generating Commit Messages. In *2015 IEEE/ACM 37th IEEE Int. Conf. on Software Engineering*, Vol. 2. IEEE, 709–712. https://doi.org/10.1109/ICSE.2015.229

[33] Mario Linares-Vásquez, Bogdan Dit, and Denys Poshyvanyk. 2013. An exploratory analysis of mobile development issues using Stack Overflow. In *Proc. of the 10th Working Conf. on Mining Software Repositories (MSR)*. IEEE, 93–96. https://doi.org/10.1109/MSR.2013.6624014

[34] Mario Linares-Vásquez, Boyang Li, Christopher Vendome, and Denys Poshyvanyk. 2016. Documenting Database Usages and Schema Constraints in Database-centric Applications. In *Proc. of the 25th Int. Symp. on Software Testing and Analysis (ISSTA 2016)*. ACM, 270–281. https://doi.org/10.1145/2931037.2931072

[35] Mario Linares-Vásquez, Kevin Moran, and Denys Poshyvanyk. 2017. Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 399–410. https://doi.org/10.1109/ICSME.2017.27

[36] Rafael Lotufo, Zeeshan Malik, and Krzysztof Czarnecki. 2012. Modelling the 'Hurried' bug report reading process to summarize bug reports. In *Proceedings of the 28th IEEE Int. Conf. on Soft. Maintenance (ICSM)*. IEEE, 430–439. https://doi.org/10.1109/ICSM.2012.6405303

[37] Senthil Mani, Rose Catherine, Vibha Singhal Sinha, and Avinava Dubey. 2012. AUSUM: Approach for Unsupervised Bug Report Summarization. In *Proceedings of the ACM SIGSOFT 20th Int. Symp. on the Foundations of Software Engineering (FSE'12)*. ACM, 11:1–11:11. https://doi.org/10.1145/2393596.2393607

[38] Paul W. McBurney, Cheng Liu, Collin McMillan, and Tim Weninger. 2014. Improving Topic Model Source Code Summarization. In *Proc. of the 22nd Int. Conf. on Program Comprehension (ICPC 2014)*. ACM, 291–294. https://doi.org/10.1145/2597008.2597793

[39] Paul W. McBurney and Collin McMillan. 2014. Automatic Documentation Generation via Source Code Summarization of Method Context. In *Proc. of the 22nd Int. Conf. on Program Comprehension (ICPC 2014)*. ACM, 279–290. https://doi.org/10.1145/2597008.2597149

[40] Paul W. McBurney and Collin McMillan. 2016. Automatic Source Code Summarization of Context for Java Methods. *IEEE Transactions on Software Engineering* 42, 2 (Feb 2016), 103–119. https://doi.org/10.1109/TSE.2015.2465386

[41] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Qing Xie, and Chen Fu. 2011. Portfolio: Finding Relevant Functions and Their Usage. In *Proc. of the 33rd Int. Conf. on Soft. Eng. (ICSE 2011)*. ACM, 111–120. https://doi.org/10.1145/1985793.1985809

[42] Naouel Moha, Yann-Gaël Guéhéneuc, Laurence Duchien, and Anne-Françoise Le Meur. 2010. DECOR: A Method for the Specification and Detection of Code

and Design Smells. *IEEE Trans. Software Eng.* 36, 1 (2010), 20–36.

[43] Kjetil Molkken and Magne Jrgensen. 2003. A Review of Surveys on Software Effort Estimation. In *Proceedings of the 2003 International Symposium on Empirical Software Engineering*. IEEE Comp. Soc., 223–230.

[44] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and Vijay Shanker. 2013. Automatic Generation of Natural Language Summaries for Java Classes. In *21st IEEE Int. Conf. on Program Comprehension (ICPC'13)*. IEEE, 23–32.

[45] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrian Marcus. 2015. How Can I Use This Method?. In *Proc. of the 37th IEEE/ACM Int. Conf. on Software Engineering (ICSE 2015)*. IEEE Press, 880–890.

[46] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2014. Automatic Generation of Release Notes. In *Proc. of the 22nd ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (FSE 2014)*. ACM, 484–495. https://doi.org/10.1145/2635868.2635870

[47] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2017. ARENA: An Approach for the Automated Generation of Release Notes. *IEEE Transactions on Software Engineering* 43, 2 (2017), 106–127. https://doi.org/10.1109/TSE.2016.2591536

[48] Sebastiano Panichella, Annibale Panichella, Moritz Beller, Andy Zaidman, and Harald C. Gall. 2016. The Impact of Test Case Summaries on Bug Fixing Performance: An Empirical Investigation. In *Proc. of the 38th Int. Conf. on Software Engineering (ICSE 2016)*. ACM, 547–558. https://doi.org/10.1145/2884781.2884847

[49] Reinhold Plösch, Andreas Dautovic, and Matthias Saft. 2014. The Value of Software Documentation Quality. In *Proc. of the 14th Int. Conf. on Quality Software*. IEEE, 333–342. https://doi.org/10.1109/QSIC.2014.22

[50] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining StackOverflow to Turn the IDE into a Self-confident Programming Prompter. In *Proc. of the 11th Working Conf. on Mining Software Repositories (MSR 2014)*. ACM, 102–111. https://doi.org/10.1145/2597073.2597077

[51] Luca Ponzanelli, Simone Scalabrino, Gabriele Bavota, Andrea Mocci, Rocco Oliveto, Massimiliano Di Penta, and Michele Lanza. 2017. Supporting Software Developers with a Holistic Recommender System. In *Proc. of the 39th IEEE/ACM Int. Conf. on Software Engineering (ICSE)*. IEEE Press, 94–105. https://doi.org/10.1109/ICSE.2017.17

[52] Sarah Rastkar, Gail C Murphy, and Gabriel Murray. 2014. Automatic Summarization of Bug Reports. *IEEE Transactions on Software Engineering* 40, 4 (2014), 366–380.

[53] Steven P. Reiss. 2009. Semantics-based Code Search. In *Proc. of the 31st Int. Conf. on Soft. Eng. (ICSE 2009)*. IEEE, 243–253. https://doi.org/10.1109/ICSE.2009.5070525

[54] Martin P. Robillard. 2009. What makes APIs hard to learn? Answers from developers. *IEEE Software* 26, 6 (2009), 27–34. https://doi.org/10.1109/MS.2009.193

[55] Martin P. Robillard and Robert Deline. 2011. A field study of API learning obstacles. *Empirical Software Engineering* 16, 6 (2011), 703–732. https://doi.org/10.1007/s10664-010-9150-8

[56] 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 *Proc. of the 33rd IEEE Int. Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 479–483. https://doi.org/10.1109/ICSME.2017.17

[57] Paige Rodeghero, Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Detecting User Story Information in Developer-Client Conversations to Generate Extractive Summaries. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 49–59.

[58] Paige Rodeghero, Collin McMillan, Paul W. McBurney, Nigel Bosch, and Sidney D'Mello. 2014. Improving Automated Source Code Summarization via an Eye-tracking Study of Programmers. In *Proc. of the 36th Int. Conf. on Software Engineering (ICSE 2014)*. ACM, 390–401. https://doi.org/10.1145/2568225.2568247

[59] Christoffer Rosen and Emad Shihab. 2016. What Are Mobile Developers Asking About? A Large Scale Study Using Stack Overflow. *Empirical Softw. Eng.* 21, 3 (2016), 1192–1223. https://doi.org/10.1007/s10664-015-9379-3

[60] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. 2009. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Sci. Comput. Program.* 74, 7 (May 2009), 470–495.

[61] S. M. Sohan, Frank Maurer, Craig Anslow, and Martin P. Robillard. 2017. A study of the effectiveness of usage examples in REST API documentation. *Proc. of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* 2017-October (2017), 53–61. https://doi.org/10.1109/VLHCC.2017.8103450

[62] Rodrigo Souza and Allan Oliveira. 2017. GuideAutomator: Continuous Delivery of End User Documentation. In *39th IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track, ICSE-NIER*. IEEE, 31–34. https://doi.org/10.1109/ICSE-NIER.2017.10

[63] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. 2010. Towards Automatically Generating Summary Comments for Java Methods. In *Proc. of the IEEE/ACM Int. Conf. on Automated Software Engineering*. ACM, 43–52.

[64] Jeffrey Stylos and Brad A. Myers. 2006. Mica: A Web-Search Tool for Finding API Components and Examples. In *Proc. of the Visual Languages and Human-Centric Computing (VLHCC 2006)*. IEEE, 195–202. https://doi.org/10.1109/VLHCC.2006.32

[65] Shin Hwei Tan, Darko Marinov, Lin Tan, and Gary T Leavens. 2012. @tComment: Testing Javadoc Comments to Detect Comment-Code Inconsistencies. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 260–269.

[66] Suresh Thummalapenta and Tao Xie. 2007. Parseweb: A Programmer Assistant for Reusing Open Source Code on the Web. In *Proc. of the 22nd IEEE/ACM Int. Conf. on Automated Soft. Eng. (ASE)*. ACM, 204–213. https://doi.org/10.1145/1321631.1321663

[67] Michele Tufano, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. 2017. There and back again: Can you compile that snapshot? *Journal of Software: Evolution and Process* 29, 4 (2017), e1838. https://doi.org/10.1002/smr.1838

[68] Michele Tufano, Fabio Palomba, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Andrea De Lucia, and Denys Poshyvanyk. 2017. When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away). *IEEE Trans. Software Eng.* 43, 11 (2017), 1063–1088.

[69] Gias Uddin and Martin P. Robillard. 2015. How API Documentation Fails. *IEEE Software* 32, 4 (2015), 68–75. https://doi.org/10.1109/MS.2014.80

[70] Todd Waits and Joseph Yankel. 2014. Continuous system and user documentation integration. In *2014 IEEE International Professional Communication Conference (IPCC)*. IEEE, 1–5. https://doi.org/10.1109/IPCC.2014.7020385

[71] Fengcai Wen, Csaba Nagy, Gabriele Bavota, and Michele Lanza. 2019. A large-scale empirical study on code-comment inconsistencies. In *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019*. IEEE Press, 53–64.

[72] Annie T. T. Ying and Martin P. Robillard. 2013. Code Fragment Summarization. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'13)*. ACM, 655–658. https://doi.org/10.1145/2491411.2494587

[73] Junji Zhi, Vahid Garousi-Yusifoglu, Bo Sun, Golara Garousi, Shawn Shahnewaz, and Guenther Ruhe. 2015. Cost, benefits and quality of software development documentation: A systematic mapping. *Journal of Systems and Software* 99 (2015), 175–198. https://doi.org/10.1016/j.jss.2014.09.042