Software Documentation Issues Unveiled

Emad Aghajani*, Csaba Nagy*, Olga Lucero Vega-Márquez†
Mario Linares-Vásquez†, Laura Moreno‡, Gabriele Bavota*, Michele Lanza*
*Software Institute, Università della Svizzera italiana (USI), Switzerland
†Systems and Computing Engineering Department, Universidad de los Andes, Colombia
‡Department of Computer Science, Colorado State University, USA

***Abstract*—(Good) Software documentation provides developers and users with a description of what a software system does, how it operates, and how it should be used. For example, technical documentation (*e.g.,* an API reference guide) aids developers during evolution/maintenance activities, while a user manual explains how users are to interact with a system. Despite its intrinsic value, the creation and the maintenance of documentation is often neglected, negatively impacting its quality and usefulness, ultimately leading to a generally unfavorable take on documentation.**

Previous studies investigating documentation issues have been based on surveying developers, which naturally leads to a somewhat biased view of problems affecting documentation. We present a large scale empirical study, where we mined, analyzed, and categorized 878 documentation-related artifacts stemming from four different sources, namely mailing lists, Stack Overflow discussions, issue repositories, and pull requests. The result is a detailed taxonomy of documentation issues from which we infer a series of actionable proposals both for researchers and practitioners.

***Keywords*-Documentation; Empirical Study**

I. Introduction

Good ~~old~~ documentation, the ideal companion of any software system, is intended to provide stakeholders with useful knowledge about the system and related processes. Depending on the target audience, the contents of documentation varies. For example, technical documentation (*e.g.,* API reference guides) describes information about the design, code, interfaces and functionality of software to support developers in their tasks, while user documentation (*e.g.,* user manuals) explains to end-users how they should use the software application.

Despite the undeniable practical benefits of documentation during software development and evolution activities [1]–[4], its creation and maintenance have been often neglected [1], [3]–[7], leading to inadequate and even inexistent documentation. These and other aspects of documentation (*e.g.,* needs, learning obstacles) have been investigated through interviews with and surveys of practitioners, with the general goal of identifying the root causes of documentation issues (*e.g.,* inaccuracy, outdatedness). To address these issues (at least partially), different approaches and tools have been proposed to aid developers during software documentation, including its automatic generation [1], [8]. For example, a recent proposal by Robillard *et al.* [9] suggests a paradigm shift towards systems that automatically generate documentation in response to a developer's query, while considering her working context.

However, to achieve high-quality automatic documentation systems, we require first a deep understanding of software practitioners' needs. Although existing studies have revealed some of these needs, their results are limited by the low number and lack of diversity of practitioners questioned and documentation artifacts analyzed.

To overcome these limitations, we qualitatively analyzed different types of artifacts from diverse data sources and identified the *issues that developers face when dealing with documentation*. Specifically, we mined open source software repositories and examined a set of 878 artifacts corresponding to development emails, programming forum discussions, issues and pull requests related to software documentation. For each artifact, we determined the reported issue, the type of documentation presenting it, and the proposed solution, as well as the documentation tools discussed. Based on our analysis, we built a *comprehensive taxonomy consisting of 162 types of documentation issues* linked to (i) the information it contains, (ii) how the information is presented, (iii) the documentation process and (iv) documentation tool support. We describe and exemplify each category of documentation issues. We also discuss their implications in software research and practice, deriving actionable items needed to address them.

II. Related Work

Software documentation has inspired research mainly on two fronts: tools & approaches and (empirical) studies.

Tools & Approaches. There has been much work on building tools to support the automated generation of documentation.

Software summarization has shown progress with different techniques and tools for generating abstractive and extractive summaries of diverse software artifacts, such as bug reports [10]–[12], classes and methods [8], [13]–[19], unit tests [20], [21], commit messages [22]–[24], release notes [25], [26], user reviews [27], code examples [28] and user stories [29]. Approaches aimed at supporting developers during coding have also been developed. Code search engines and recommendation systems are available for retrieving API and code usage examples [30]–[33], code fragments implementing specific features [34]–[37] and crowd knowledge [38], [39].

Despite these efforts, automated documentation is still wishful thinking. A first research road map to enable automated on-demand documentation has however been drawn by the recent proposal of Robillard *et al.* [9].

TABLE I
SUMMARY OF PREVIOUS STUDIES ON SOFTWARE DOCUMENTATION ASPECTS.

Study	**Artifacts**	**Summary of findings** (related to concerns and quality attributes)
Forward and Lethbridge (2002) [1]: Questionnaire with 48 participants (from sw industry, research peers, and mail lists members).	Software documentation regularly used by participants	Despite documentation being outdated, practitioners learn how to deal with it. *"Software documentation tools should seek to better extract knowledge from core resources. Software professionals value technologies that improve automation of the documentation process, and its maintenance."*
Kajko-Mattsson (2005) [3]: Exploratory study with 18 Swedish organizations.	Maintenance-related documentation artifacts	*"Documentation within corrective maintenance is still a very neglected issue."*
Chen and Huang (2009) [6]: Questionnaire with 137 project managers and sw engineers of the Chinese Information Service Industry Association of Taiwan.	Software documentation regularly used by participants	Most typical problems in software documentation quality for maintenance are that software documentation is *untrustworthy, inadequate, incomplete or does not even exist, lacks traceability, does not include its changes, and lacks integrity and consistency*.
Robillard (2009) [40]: Personal interviews with 80 professionals at Microsoft.	API documentation and source code	The top obstacles for API learning are: *resources for learning* (documentation, examples, *etc.*), *API structure, Background, Technical environment, Process.* API documentation must *include good examples, be complete, support complex usage scenarios, be organized, and have better design*.
Dagenais and Robillard (2010) [41]: (i) A qualitative study with 12 contributors and 10 users of open source projects, and (ii) an evolution analysis of 19 documents from 10 open source projects.	Open source projects documentation in a repository or wiki (*e.g.,* Django, Firefox and Eclipse)	In open source projects, knowing the relationships between documentation and decisions of contributors help to define better techniques for documentation creation and maintenance. When a wiki is selected to host documentation, its quality is threatened by erroneous updates, SPAM or irrelevant content (URLs included). This requires more effort for maintaining wikis.
Robillard and Deline (2011) [42]: (i) An initial questionnaire, (ii) a set of qualitative in-person interviews and (iii) a questionnaire with 440 developers at Microsoft.	API documentation regularly used by participants	Relevant issues in the documentation that affect the developers learning experience: *"documentation of intent, code examples, cookbooks for mapping usage scenarios to API elements, penetrability of the API, and format and presentation of the documentation."*
Plösch et al. (2014) [43]: Online questionnaire with 88 software professionals, mainly German speakers.	Software documentation regularly used by participants	The most important attributes are accuracy, clarity, consistency, readability, structuredness and understandability. *"There is a need for automatic analysis of software documentation quality."*
Zhi et al. (2015) [4]: Mapping study about a set of 69 papers from 1971 to 2011.	N/A	Documentation quality attributes that appear in most of the papers are *completeness, consistency and accessibility*. More empirical evidence is required involving large-scale development projects, or larger samples of participants from various organizations; more industry-academia collaborations are also required, and more estimation models or methods to assess documentation.
Garousi et al. (2015) [44]: Industry case study with analysis of documentation (using the taxonomy by Zhi [4]) and a questionnaire with 25 employees of NovAtel Inc.	Source code and a sample of software documentation (design, tests and processes)	Technical documentation is preferred during development than during maintenance tasks; the preferred source of information for maintenance is the source code; other sources of information have no significant impact on developers' preferences.
Uddin and Robillard (2015) [45]: A case study and a questionnaire with 230 software professionals from IBM.	API documentation	The top 10 problems in API documentation are (i) incompleteness, (ii) ambiguity, (iii) unexplained examples, (iv) obsoleteness, (v) inconsistency and (vi) incorrectness; while in presentation are (vii) bloat, (viii) fragmentation, (ix) excess structural information and (x) tangled information.
Alhindawi et al. (2016) [46]: Topic-modeling-based study.	KDE/KOffice source base and its external documentation	A novel approach for evaluating documentation quality. Tools that can automatically assess the software documentation quality in an are highly demanded. Labeling and grouping documentation would impact its quality positively.
Sohan et al. (2017) [47]: Controlled study with 26 software engineers.	WordPress REST API documentation	Developers feel more satisfied when having examples. When documentation lacks examples, developers spend more time on coding, execute more trial attempts, and have lower success rates.

(Empirical) Studies. Software documentation has been analyzed in diverse empirical studies that (i) report evidence of its importance and impact in the software life cycle [1], [3], [6], [41], [42], [44], (ii) describe problems that developers face when dealing with it [3], [6], [40], [42], [45], (iii) list quality attributes required in documentation [42], [44], [45], [48], [49], (iv) provide recommendations for constructing it (including standards) [1], [3], [40]–[42], [44], [45], [50], [51], and (v) propose frameworks and tools for evaluating documentation concerns such as cost, benefit and quality attributes [44], [46]–[49]. Due to space limitations we summarize the closest ones to our study in Table I.

The mapping study by Zhi *et al.* [4] is notable, as it reviews about 100 documentation-related papers and reports that the most frequently discussed quality attributes are *completeness, consistency and accessibility*. Zhi *et al.* conclude that software documentation is an immature area, and that stronger empirical evidence is needed to gain a deeper understanding of it.

Most of the aforementioned studies gathered information directly from participants and used practitioner samples restricted to a specific context (*e.g.,* a company). These studies are therefore not diverse enough in terms of analyzed artifacts and programming languages used by developers, and the largest samples reported in the studies are 440 (Robillard and Deline [42]) and 230 practitioners (Uddin and Robillard [45]). To avoid some of the limitations imposed by interviews and surveys, we opted for an approach that allowed us to study a wider population in terms of number and types of artifacts by mining different data sources.

Our results complement previous categorizations of documentation issues with a taxonomy that considers documentation content, processes and tools.

Ours is the first mining-based study focused on identifying documentation issues as discussed by practitioners in software repositories. Previous studies following a mining-based strategy are more general, identifying topics discussed by developers [52]–[54] or by apps' users [55].

III. EMPIRICAL STUDY DESIGN

Our goal is to answer the following research question (RQ): *What are the documentation issues faced by developers?*

A. Data Collection

Our data collection consists of two steps. First, we adopt an automatic process based on keyword matching to mine candidate artifacts related to documentation from the four analyzed sources (*i.e.,* emails, issues and pull requests of open source projects, and Stack Overflow threads). Then, we manually analyze a statistically significant sample of artifacts to categorize them based on the issues they discuss, the solutions they propose, and the type of documentation they involve.

1) Identification of Candidate Artifacts Related to Documentation Issues: Table II summarizes the artifacts automatically collected from the four sources (see column "candidate artifacts"). We discuss the process adopted in each case.

TABLE II
STUDY DATASET

Source	**Candidate Artifacts**	**Manually Analyzed**	**False Posit.**	**Valid Artifacts**	**Labeled Sentences**
Issues	394,504	345	19	324	562
Mailing Lists	6,898	101	5	95	220
Pull Requests	375,745	332	21	310	581
Stack Overflow	28,792	100	4	95	185
Overall	**805,939**	**878**	**49**	**824**	**1,548**

Stack Overflow (SO). We mined from the official SO dump of June 2018 all discussions having a question labeled with a documentation-related tag. To determine these tags, we searched for all tags related to documentation and documentation tools in the SO tag page by using the keywords *doc*, *documentation* and *documentor*. The latter term is known to be part of the name of tools supporting software documentation. One author then inspected all the tags resulting from these three searches to identify the ones actually related to software documentation and/or documentation tools. During the inspection, the author read the tag name, the tag description and some of the questions in which the tag was used. This process resulted in the selection of 23 tags (*e.g.,* code-documentation, phpdocumentor, design-documents) that were used to search for the related discussions in SO. The first 30 results (discussions) returned by the 23 searches were manually inspected to look for additional documentation-related tags missed in the first step. The process was iterated with the newly founded tags until no new tags were found in the top 30 results of the tag searches. This resulted in a total of 78 (23+55) documentation-related tags (available in our replication package [56]).

Next, we queried the SO dump to extract all discussions having a question with a non-negative score and tagged with one or more of the relevant 78 tags. We removed questions with a negative score to filter out irrelevant discussions. This process resulted in the selection of 28,792 discussions. For each of them, we kept the question, the two top-scored answers and the accepted answer (if any).

GitHub Issues and Pull Requests. We downloaded the GitHub Archive [57] containing every public GitHub event occurring between January 2015 and April 2018. While older data is available, we excluded it since some of the information needed for our study was only archived starting from 2015. We extracted all events of type *IssuesEvent*, *IssueCommentEvent*, *PullRequestEvent* and *PullRequestReviewCommentEvent*.

These events capture the opening/closing of issues and pull requests as well as all the discussion held for them through comments. A detailed description of these event types is available online [58]. Then, we selected issues and pull requests from projects having at least ten forks and/or stars to exclude "toy" projects. Finally, we adopted a keyword-matching approach to extract issues and pull requests related to documentation. We started from the 78 SO tags previously mentioned and *converted* them into 56 "keywords". This means, for example, that the SO tag *design-documents* was converted into *design doc* (to match "design document", "design documents" and "design doc"), while tags including the word "documentation" (*e.g., xml-documentation*) were replaced with the keyword *documentation*, since matching this keyword will also match the more specific ones. We also added 11 keywords that we considered relevant but were not derived from any of the 78 SO tags. For example, while the keyword *api doc* was derived from the SO tag *api-doc*, we also added *api manual*. In total, we defined 66 documentation-related keywords [56].

We extracted all the issues/pull requests having at least one of the 66 keywords in their title and/or in their first post (*i.e.,* the one opening the issue or the pull request). This resulted in the selection of 394,504 issues and 375,745 pull requests.

Mailing Lists. We built a crawler to mine the mail archives of the Apache Software Foundation (ASF) [59]. The ASF archives all emails exchanged in the mailing lists of the projects it runs. Each of its 295 projects has several mailing lists focused on different topics. We mined all mailing lists named *docs* (discussions related to documentation), *dev* (discussions among developers) and *users* (discussions involving both users and developers), for a total of 480 mailing lists. For the threads extracted from the *docs* mailing lists, we did not apply any filter. For the threads extracted from the *dev* and the *users* mailing lists, we only selected those containing in the subject at least one of 66 documentation-related keywords we previously defined. This resulted in the extraction of 6,898 email threads, each one composed by one or more messages.

2) Manual Classification of Documentation Issues: Once we collected the candidate artifacts, we manually analyzed a statistically significant sample ensuring a 99% confidence level $\pm$ 5%. This resulted in the selection of 665 artifacts for our manual analysis, out of the 805,939 artifacts collected from the four sources. Since the number of collected artifacts is substantially different between the four sources (Table II), we decided to randomly select the 665 artifacts by considering these proportions. A simple proportional selection would basically discard SO and mailing lists from our study, since issues and pull requests account for over 90% of our dataset. Indeed, this would result in the selection of 311 pull requests, 326 issues, 24 SO discussions and 6 mailing list threads. For this reason, we adopted the following sampling procedure: for SO and mailing lists, we targeted the analysis of 96 artifacts each, ensuring a 95% confidence level $\pm$ 10% within those two sources. For issues and pull requests, we adopted the proportional selection as explained above. This resulted in 829 artifacts to be manually analyzed (99% confidence $\pm$ 4.5%).

The selected artifacts were manually analyzed by six of the authors with the goal of classifying them as false positive (*i.e.,* unrelated to documentation issues) or assigning a set of *labels* describing (i) the documentation issue discussed, (ii) the solution proposed/adopted, (iii) the type of the documentation and (iv) the documentation tools discussed. Each of these labels was optional. For example, it is possible that only the issue type and the solution were labeled for an artifact.

For two of the four categories, namely issue type and documentation type, we started from a predefined list of labels. For the issue types, we used the 13 quality attributes defined by Zhi *et al.* [4]. For the type of documentation, we had 11 predefined labels that we selected based on our experience (*e.g.,* code comments). See [56] for the list of predefined labels.

The labeling was supported by a Web app that we developed for this task and for conflict resolution. Each author independently labeled artifacts randomly assigned to her by the Web app, selecting a proper label among the predefined ones or defining a new label when needed. To assign a label, the author inspected the whole artifact and, in the case of issues and pull requests, also the related commits. Every time an author had to label an artifact, the Web app also showed the list of labels created by all taggers so far. The labeling was performed at **sentence level**: The Web app allowed the author to select one sentence from the artifact at a time and assign labels to it. This means that multiple sentences could be labeled for each artifact and hence multiple labels could be assigned to it. This allowed us to create a database (publicly available [56]) of labeled sentences related to documentation artifacts.

Each artifact was assigned to two authors by the Web app. In case both of them classified the artifact as a false positive, the artifact was replaced with another one randomly selected from the same source (*e.g.,* a false positive email thread was replaced with another email thread). For each artifact X in which there was a conflict in the assigned labels, a third author (not previously involved in the labeling of X) was assigned to solve it. A conflict in this scenario can happen for many reasons. First, the two authors could label different sentences in the artifact. Second, assuming the same sentences are selected, different "categories" of labels could be assigned to the sentences (*e.g.,* one author labels a sentence as discussing the issue, one as presenting a solution). Third, assuming the same sentences and the same categories of labels are selected, the label values differ (*e.g.,* different solutions indicated for the same sentence). Fourth, one author could classify the artifact as a false positive, while the other could label it. For these reasons, we had a high number of conflicted artifacts (765 out of 829 — 92.27%). We solved some specific cases automatically. In particular, if two authors (i) labeled for the same artifact two different sentences S_i and S_j where S_i is a substring of S_j (or *vice versa*), and (ii) had no conflicts between the label values, we automatically solved the conflict by selecting the longest sentence as the valid one. This reduced the number of conflicted artifacts to 592, which were manually reviewed by a third author who could accept a conflicting sentence (and apply minor modifications if necessary) or discard it.

In this final process, 5 artifacts were discarded as false positives. The final number of sentences labeled for each type of artifact is reported in Table II.

B. Data Analysis

We answer our RQ by presenting a taxonomy of the types of documentation issues found in our analysis.

Such a taxonomy was defined in an open discussion involving all the authors and aimed at merging similar labels and hierarchically organizing them (see Figure 1). We focus our qualitative analysis on specific categories of issue types. For each category, we present interesting examples and common solutions, and discuss implications for researchers and practitioners.

C. Replication Package

All the data used in our study is publicly available [56].

IV. Results Discussion

As a result of the labeling process, we obtained 824 artifacts including a total of 1,548 labeled sentences.

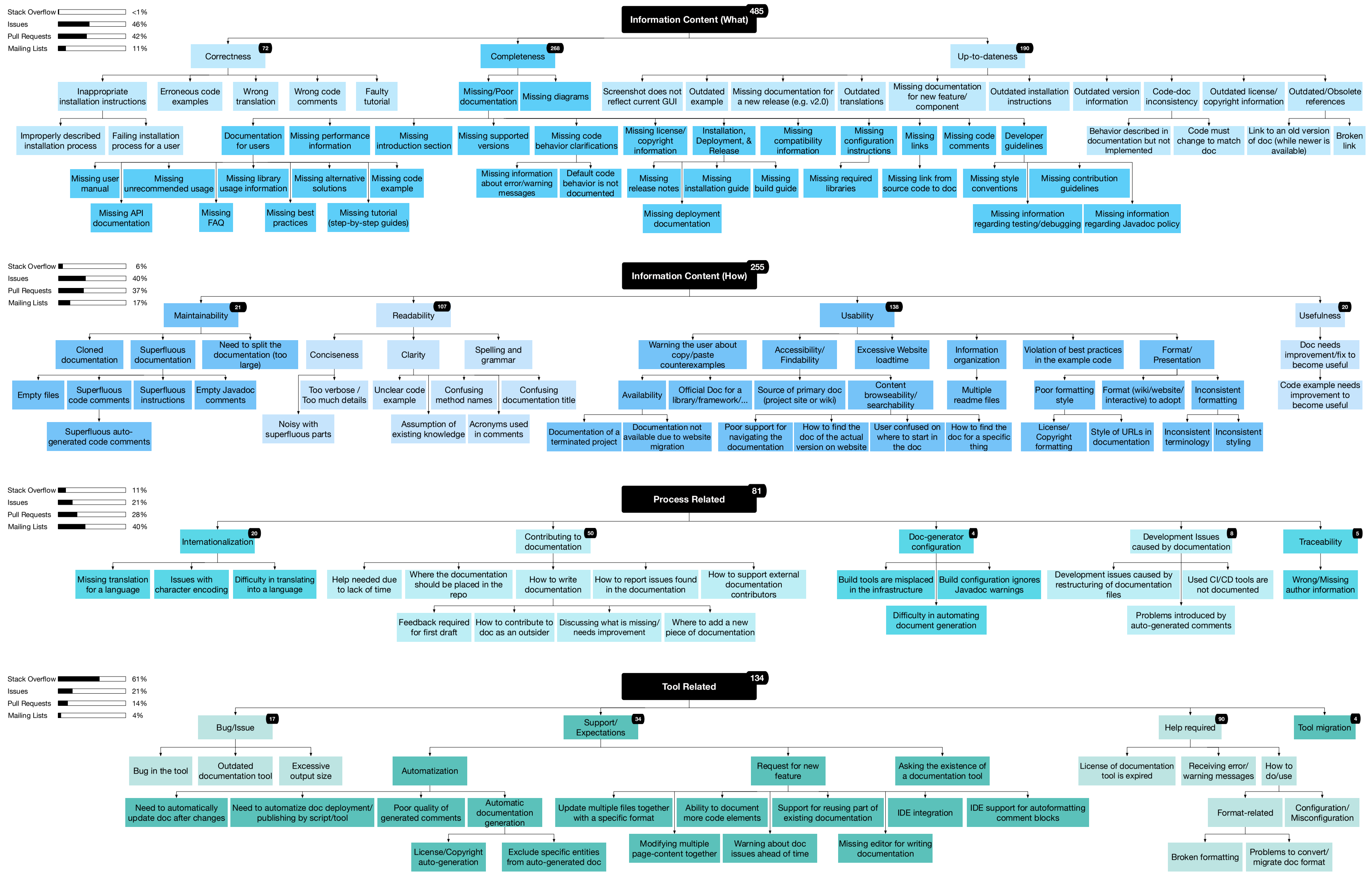
Figure 1 shows the hierarchical taxonomy of the 162 documentation issue types that we identified. They are grouped into four main categories: (i) problems related to the *information content* of the documentation describe issues arising from "what" is written in the documentation; (ii) issues classified under the *information content (how)* category focus on "how" the content is written and organized; (iii) the *process-related* category groups issues related to the documentation process; and (iv) *tool-related* matters originate from the usage of a documentation tool. The number shown in the main categories of Figure 1 represents the number of artifacts related to that issue (*e.g.,* 81 artifacts were related to process-related issues). Note that a single artifact might discuss multiple types of issues. Figure 1 also shows the distribution of the analyzed artifacts among the four sources we analyzed. Interestingly, problems related to the content and how it is presented/organized are mostly discussed in issues and pull requests; and discussions about the documentation process and tools-related issues are mainly held in mailing lists and SO respectively.

For each category, we next describe representative examples and discuss implications for researchers (indicated with the 🧪 icon) and/or practitioners (🔧 icon) derived from our findings.

A. Information Content (What)

A total of 485 artifacts discuss issues related to the information content, *i.e.,* "what" is written in the documentation.

Correctness (72). Correct documentation provides accurate information in accordance with facts [4]. Incorrect documentation might have unforeseen serious consequences, going beyond wasted time trying to replicate a wrong code example or following the wrong steps in a tutorial. This is the case of an issue filed for the *acid-state* project, a tool providing ACID guarantees to serializable Haskell data structures. As reported in the issue, a false claim in the documentation could lead to data loss: *"This could easily cause permanent data loss if the user then proceeds to remove the Archive folder, which is claimed to be safe by the documentation"* [65].



Information Content (What) 485
Stack Overflow <1%
Issues 46%
Pull Requests 42%
Mailing Lists 11%
Correctness 72
Completeness 268
Up-to-dateness 190
Inappropriate installation instructions
Erroneous code examples
Wrong translation
Wrong code comments
Faulty tutorial
Improperly described installation process
Failing installation process for a user
Missing/Poor documentation
Missing diagrams
Documentation for users
Missing performance information
Missing introduction section
Missing supported versions
Missing code behavior clarifications
Missing license/ copyright information
Installation, Deployment, & Release
Missing compatibility information
Missing configuration instructions
Missing links
Missing code comments
Developer guidelines
Missing user manual
Missing unrecommended usage
Missing library usage information
Missing alternative solutions
Missing code example
Missing API documentation
Missing FAQ
Missing best practices
Missing tutorial (step-by-step guides)
Missing information about error/warning messages
Default code behavior is not documented
Missing release notes
Missing installation guide
Missing build guide
Missing required libraries
Missing link from source code to doc
Missing style conventions
Missing contribution guidelines
Missing deployment documentation
Missing information regarding testing/debugging
Missing information regarding Javadoc policy
Screenshot does not reflect current GUI
Outdated example
Missing documentation for a new release (e.g. v2.0)
Outdated translations
Missing documentation for new feature/ component
Outdated installation instructions
Outdated version information
Code-doc inconsistency
Outdated license/ copyright information
Outdated/Obsolete references
Behavior described in documentation but not Implemented
Code must change to match doc
Link to an old version of doc (while newer is available)
Broken link
Information Content (How) 255
Stack Overflow 6%
Issues 40%
Pull Requests 37%
Mailing Lists 17%
Maintainability 21
Readability 107
Usability 138
Usefulness 20
Cloned documentation
Superfluous documentation
Need to split the documentation (too large)
Empty files
Superfluous code comments
Superfluous instructions
Empty Javadoc comments
Superfluous auto-generated code comments
Conciseness
Clarity
Spelling and grammar
Too verbose / Too much details
Noisy with superfluous parts
Unclear code example
Confusing method names
Confusing documentation title
Assumption of existing knowledge
Acronyms used in comments
Warning the user about copy/paste counterexamples
Accessibility/ Findability
Excessive Website loadtime
Information organization
Violation of best practices in the example code
Format/ Presentation
Availability
Official Doc for a library/framework/...
Source of primary doc (project site or wiki)
Content browseability/ searchability
Multiple readme files
Poor formatting style
Format (wiki/website/ interactive) to adopt
Inconsistent formatting
Documentation of a terminated project
Documentation not available due to website migration
Poor support for navigating the documentation
How to find the doc of the actual version on website
User confused on where to start in the doc
How to find the doc for a specific thing
License/ Copyright formatting
Style of URLs in documentation
Inconsistent terminology
Inconsistent styling
Doc needs improvement/fix to become useful
Code example needs improvement to become useful
Process Related 81
Stack Overflow 11%
Issues 21%
Pull Requests 28%
Mailing Lists 40%
Internationalization 20
Contributing to documentation 50
Doc-generator configuration 4
Development Issues caused by documentation 8
Traceability 5
Missing translation for a language
Issues with character encoding
Difficulty in translating into a language
Help needed due to lack of time
Where the documentation should be placed in the repo
How to write documentation
How to report issues found in the documentation
How to support external documentation contributors
Feedback required for first draft
How to contribute to doc as an outsider
Discussing what is missing/ needs improvement
Where to add a new piece of documentation
Build tools are misplaced in the infrastructure
Build configuration ignores Javadoc warnings
Difficulty in automating document generation
Development issues caused by restructuring of documentation files
Used CI/CD tools are not documented
Problems introduced by auto-generated comments
Wrong/Missing author information
Tool Related 134
Stack Overflow 61%
Issues 21%
Pull Requests 14%
Mailing Lists 4%
Bug/Issue 17
Support/ Expectations 34
Help required 90
Tool migration 4
Bug in the tool
Outdated documentation tool
Excessive output size
Automatization
Request for new feature
Asking the existence of a documentation tool
Need to automatically update doc after changes
Need to automatize doc deployment/ publishing by script/tool
Poor quality of generated comments
Automatic documentation generation
License/Copyright auto-generation
Exclude specific entities from auto-generated doc
Update multiple files together with a specific format
Ability to document more code elements
Support for reusing part of existing documentation
IDE integration
IDE support for autoformatting comment blocks
Modifying multiple page-content together
Warning about doc issues ahead of time
Missing editor for writing documentation
License of documentation tool is expired
Receiving error/ warning messages
How to do/use
Format-related
Configuration/ Misconfiguration
Broken formatting
Problems to convert/ migrate doc format

Fig. 1. Documentation Issues Taxonomy

The type of documentation most frequently impacted by correctness issues was code examples (*e.g.,* [66]), accounting for 50% of the cases in which we labeled a documentation type, followed by installation guidelines (20%, *e.g.,* [66]). Correctness issues in code examples include syntactic mistakes (*e.g., "the documentation gives the following example [...] but running it leads to ERROR: syntax: [...] is not a symbol"* [67]), as well as more serious programming errors (*e.g., "one of the example fixture files in the documentation would not work because it contains references to objects that have not yet been declared"* [68]). In general, due to their potential consequences, correctness issues were handled with care by developers. For example, we found a case of correctness issue caused by a wrong translation, where developers not only fixed the mistranslation but also decided to have the document reviewed by a native speaker [69].

Completeness (268). Documentation is incomplete if it does not contain the information about the system or its modules needed by practitioners/users to perform their tasks [4].

Completeness accounts for 55% of the issues related to the documentation content. We observed different causes of incompleteness. For example, in an email sent to the *Apache httpd* mailing list, a user complained about missing definitions of ambiguous terms: *"is there any idea what "frequently" might mean?"* [70]. Indeed, the documentation states *"[...] should result in substantial performance improvement for frequently-requests files"*, without providing a clear definition of what "frequently-requests files" are. Other common completeness issues are related to missing descriptions of library components (*e.g., "[...] missing information about the toolbar buttons"* [71]), missing API usage clarifications (*e.g., "I think that we should add documentation ensuring that the user passes a tree with reset bounds"* [72]) and lack of compatibility information (*e.g., "Explicitly mention if clang 4.x, 5.x are supported"* [73]).

API references and code comments are the types of documentation mostly affected by completeness issues.

Up-to-dateness (190). A document is outdated when it is not in sync with other parts of a system. Up-to-dateness differs from "Correctness" and "Completeness" in that the information was correct and complete before a change was introduced.

Up-to-dateness problems account for 39% of issues related to documentation content. In 21% of these cases, the inconsistency appeared to be between a system's behavior and its description in the documentation. The discrepancy was usually triggered by a change in the code that required to change parts of the documentation or to add/remove content. This latter case typically happened when new features are implemented, *e.g., "include documentation around the new field converter feature"* [74]. In other cases, instead, users complained about the documentation of a behavior that became unavailable (*e.g., "the setLeftScale and setRightScale routines mentioned in the doxygen documentation seem not to exist"* [75]).

While most of the times is the documentation that does not reflect what is implemented in the code, in other cases it is the code that needs to be updated to match the documentation. For instance, implementing a method in a non-thread-safe manner was questioned by a user as *"the callback is not thread-safe which it has to be according to the documentation"* [76].

There were also situations in which there was a debate to decide whether the code or the documentation needed adjustment to fix the inconsistency. This was the case of a GitHub issue related to an inconsistency between the documented and actual behavior of an API: *"Is this an error in the code, or an error in the documentation?"* [77].

Referring to deprecated information is another reason for up-to-dateness issues and can affect the documentation in different ways. It includes having deprecated information in the project's website (*e.g., "homepage recommends deprecated commands"* [78]), outdated copyright information [79] and version numbers [80] in the code base, as well as outdated references (*e.g.,* links to old versions of the system), which was the most prevalent issue within this category. For example, a user reported that *"the example linked in the documentation is using the 3.x version of the API, and that may be confusing to readers"* [81].

Some developers adopted preventive solutions to ensure documentation up-to-dateness, adding this as one of the items to check in the contribution to-do list [82], or even making Javadoc update mandatory for pull request acceptance [83].

Discussion and Implications. Our results highlight frequent issues related to the correctness, up-to-dateness and completeness of the information reported in the documentation. The documentation types most frequently affected by correctness issues are, not surprisingly, code examples. Indeed, as it happens with production code, bugs can affect code examples as well. A recommendation to mitigate this problem is to apply testing techniques on code examples as done on production code. However, this might not be trivial since documentation often reports incomplete examples rather than entirely runnable programs (*e.g.,* a snippet of code on how to use an API is shown, but the snippet cannot be actually compiled and run).

Devising approaches to (i) test complete/incomplete code examples in documentation and (ii) validate the consistency between snippets and source code is a research challenge for the software engineering community. Assuming the availability of such techniques, regression testing of code snippets could be performed to ensure they are always up-to-date. A more challenging scenario is to automatically generate code examples to be included in documentation.

Up-to-dateness and completeness issues can also benefit from careful traceability of information between documentation and code. We observed issues related to documented code that does not exist in the system anymore. To address this issue, on the one side, developers should keep track of documented/undocumented code components. One way of doing so is to use a contingency matrix, where rows represent code components and columns represent existing documentation artifacts. A check in the entry$_{i,j}$ would indicate that the component i is documented in the artifact j. This matrix can then be queried to check for inconsistencies. On the other side, researchers should continue their work on traceability link recovery [60], investing in the implementation of tools that can be easily adopted by developers.

Finally, some of the issues we observed (*e.g.,* ambiguous terms in the documentation) highlighted ⑂ the importance of including documentation users in the loop. Indeed, information that might look clear from the developers' perspective is not always easy to digest by the users of the system. Involving them in the review of the documentation might help in minimizing the users' learning curve and in avoiding misunderstandings.

B. Information Content (How)

A total of 255 artifacts discuss issues related to *how* the content of the documentation is written and organized.

Usability (138). Usability of documentation refers to the degree to which it can be used by readers to achieve their objectives effectively. This category covers issues affecting users' experience with the documentation.

Half of the issues (50%) were related to information findability, *i.e.,* when the desired information was available but couldn't be found by a user, *e.g., "I cannot find the description or implementation notes"* [84] or *"I can't seem to find the API documentation anywhere. Could you please host it somewhere or point me there"* [85]. Developers often handle these issues by providing users with pointers to the documentation needed. In some cases, they go further to improve the user experience by implementing a search feature in the project's website [86] or by adding more intra-documentation links [87].

Information organization (18%), *i.e.,* how intuitively and clearly the information is organized [4], constitutes the second most common concern in this category. Placing documentation in standard locations is an effective practice to help users locating it, *e.g., "the consolidated document [...] is compiled into the 'docs/' folder, because as you already said, this location is much more prominent and easier to find"* [88]. Moreover, leveraging intra-documentation links for easier navigation [88], preparing a template (*e.g., "I have setup a page template that can be used as starting point for new pages"* [89]) and adding a 'Table of Contents' for easier navigation [90] were among other popular solutions to ensure a good information organization.

Poor or inconsistent formatting was another common issue, though not really a barrier for using documentation (*e.g., "heading styles should be improved to have a better separation between H1, H2"*). In general, users only complain about formatting when other types of usability issues emerge (*e.g., "I never find what I want on revapi.org [...] the link structure is counter-intuitive, some links are somehow hidden"* [91]). We also observed intra-documentation consistency issues (*e.g., "inconsistent title between sidebar and article"* [92]) and problems related to poor content organization (*e.g., "the order of the modules on modules.html is pretty arbitrary"* [93]).

Availability, *i.e.,* whether the documentation is accessible, was also an issue in the analyzed artifacts. For example, in response to a user who was looking for the documentation of a plugin, a developer answered *"unfortunately, at the moment not all documentations for all plugins have been migrated yet. This is currently under going"* [94]. In another case where a user was looking for documentation of a terminated project [94], web archive services (*e.g.,* Wayback Machine) were suggested.

Maintainability (21). This category concerns issues related to the maintenance of documentation, *e.g.,* how easy it is to apply changes or corrections to it. Just like in source code, duplicated content caused troubles for documentation maintainers, which were mostly resolved by replacing the clone with links and references. However, we noticed that documentation frameworks often make it hard to avoid duplicates. For instance, due to the document format requirements of *Jazzy*, a documentation tool for Swift and Objective-C, users have to create unwanted duplicates for *Xcode Quick Help* (an IDE feature for showing methods' comments), as a developer reports: *"duplicating the documentation is admittedly annoying, but that's still the only thing that satisfies Quick Help"* [95]. In another scenario, due to format mismatch between GitHub pages and *AsciidoctorJ*, a Java documentation tool, the documentation content was kept in two locations: *"the reason for these 2 locations is that GitHub does not resolve the includes"* [88].

Another noticeable issue was the existence of superfluous files that might cause confusion. This was suggested by a developer of the *Apache httpd* project: *"get rid of these no-content files so they don't confuse the issue of what still needs to be documented"* [96].

Readability (107). Readability is the extent to which documentation is easy to read. Issues related to lack of clarity represented more than half (55%) of these problems. A user of the *Apache stdcxx* project complained: *"we were able to solve the problem using the information in the users guide, but [...] the documentation is rather confusing on exactly how this needs to be set"* [97]. Abstract [98], too technical [99] and too verbose/noisy [99] information were among the main reasons for poor readability. Developers reacted to these issues by rewriting unclear parts of the documentation, *e.g., "this pull request aims to better explain the differences between these two options"* [100] or *"I don't know if this is the best wording, but I found this behavior confusing and not clearly explained in the docs. Hope this clarifies things a bit"* [101].

The second most frequent cause for readability issues were simple typos. Fixing such errors was always welcome: *"language corrections would be a hugely appreciated contribution too"* [98], especially when they affected user documentation: *"[...] we do not have to be as stringent as we have to be for user visible docs"* [102].

Usefulness (20). A document is useful if it is of practical use to its readers, *i.e.,* readers can successfully achieve their goals with the help of the document. Depending on the reader's goal, usefulness can be affected by several factors. For instance, in response to a documentation update, the owner of a project suggested: *"it would be good to make this example a bit more realistic"*. In this case, the code example is neither outdated or wrong, but it required improvements to be more useful.

Many maintainers addressed usefulness by asking users' feedback on the documentation. In one scenario, developers collected user feedback to improve the documentation website in two steps: first they conducted a survey prior to documentation refactoring, and then they gathered feedback to ensure that the changes met the users' needs: *"we did a survey prior* *to the doc lockdown to get an idea of what we should focus on. Now we have a yes/no style survey to ensure that we met the user needs when it came to improving the docs"* [103].

Discussion and Implications. Besides the information content (*i.e.,* what is in the documentation), the way it can be consumed (*i.e.,* how effectively its content can be exploited) strongly influences documentation quality. As our analysis reveals, a major part of the discussed issues is related to the usability of the documentation, stemming from poor information organization and findability issues (if existing documentation cannot be found it conceptually does not exist).

Developers can prevent/address these issues by: (i) providing a search engine in the project's website to improve content findability; (ii) adopting a consistent documentation format, *e.g.,* a template that ensures the presence of intra-documentation links and a table of contents, or a style similar to existing documentation recognized by its quality (*e.g.,* see the MongoDB documentation at https://docs.mongodb.com); and (iii) archiving the documentation of old, dismissed versions of their projects in a specific location, to make them available to users who cannot update to newer versions. We also support the idea adopted by some developers to survey their users [103] to investigate their documentation needs.

In this context, researchers can contribute to improving documentation quality by working in two directions. First, in the same way that code clone detection techniques have been implemented [61], approaches to detect documentation clones and automatically remove (refactor) them could help developers in reducing redundancy in documentation. Second, similar to code readability metrics [62], readability metrics tailored for documentation could help developers in spotting and fixing readability issues. While one may think that software documentation consists only of text and, as such, standard readability metrics for text can be used (*e.g.,* the Flesch-Kincaid readability formulas [63]), software documentation is often a mix of text and code that uses domain-specific terms. Moreover, while part of the problem is to classify a document as highly/poorly readable, a more difficult challenge is to indicate to the developer the exact section of the documentation causing the readability issue. Thus, creating "documentation linters" is an interesting avenue for future research.

In addition, studies investigating the users' behavior when looking for documentation could help to define better practices for the organization and presentation of documentation.

C. Tool Related

In this section we discuss four types of issues related to documentation tools (*e.g.,* Javadoc) found in 134 artifacts.

Bug/Issue (17). This category refers to problems presented by documentation tools (*e.g.,* bugs, malfunctions) that are not originated from improper usage or configuration.

Bugs in software systems are quite common, and documentation tools are no exception.

Users often asked questions on SO when they were not sure whether they experienced a problem originating from wrong usage or a bug in the tool. These stories usually ended up in the issue tracker. In one case, when a user failed to parse a markdown file with *Doxygen* asked a question on SO saying *"By this definition, this should work [...] Is this a bug, or am I not doing it right?"*. She got a prompt response: *"This appears to be a bug in the Markdown parser you are using. You might consider reporting it to the developers of that project"*.

In another case [104], a user noticed that *stack haddock*, a toolset for Haskell development, does not generate documentation for the dependencies of the executable or test components, but only for library components. Although the issue is still unsolved and might look like a feature request, it is labeled as "Should" by developers, which implies that the current behavior needs to be changed.

Support/Expectations (34). This category covers developers' needs that were not fulfilled by documentation tools.

Users often wanted popular tools to be available in several contexts/languages, *e.g., "Is there anything like GhostDoc for C++"* [105], or *"Is there any tool which provides these Doxygen-style features for Ruby?"* [106].

New features for existing tools were requested several times, as in a scenario where a user needed quick access to Android documentation from Android Studio [107].

Automatization was also frequently discussed. A representative example is the automatic deployment of documentation, which was implemented in a project after a developer's suggestion to automatically publish the latest documentation for a specific branch [108].

Help required (90). This category covers issues caused by improper tool usage or configuration rather than by bugs.

Warning and error messages from documentation tools were discussed in all analyzed sources. Examples of these warnings/errors were related to the use of a wrong Python version [109], a wrong path in the documentation build configuration [110], inconsistencies in the Javadoc comments format [111] and empty Javadoc code comments [112]. Providing tool usage examples was the most prevalent solution (*e.g.,* [113]).

"*How to*" questions related to documentation tools, *e.g., "how can I get Sphinx to recognize type annotations?"* [109], account for 83% of the observed issues in this category. This type of question was mostly asked in SO (79%). Formatting was a major sub-issue. In one case, *phpDocumentor* generated files with a wrong format: *"When running phpDocumentor, the resulting files/ folder looks extremely weird"*. This issue has been open since November 2015 [114].

Tool migration (4). This category refers to issues related to migration, either to a newer tool version or to another tool.

Errors after migrating to a newer version of the same tool were observed in two out of four discussions we analyzed. In one scenario, a user faced numerous errors with a newer version of Javadoc: *"javadoc is having troubles compiling Tools and I can't see why. It has only happened since I migrated to Java 8. I never saw this issue with Java 7"* [115].

In another case, developers noticed that a navigation bar disappeared from the documentation after migrating to a newer *Sphinx* version. The answer noted that this was a change in the default theme, but could be set to behave as it did before.

Discussion and Implications. Most of the tool-related issues we identified can be generalized to issues experienced by users with any type of tool, not just with the one related to documentation. The prevalence of "*how to*" questions in this category, reinforce our findings on completeness and findability of documentation (see Section IV-A). Indeed, these questions are likely the result of missing (or difficult to find) documentation in documentation tools. Thus, the same previously distilled implications for researchers and practitioners apply here.

We identified many artifacts discussing feature requests or tool expectations, which carry a message for practitioners to pay attention to common needs, such as support for IDE integration, handling of multiple documents together and automatic document/comment generation.

In addition, researchers could develop approaches to help users in understanding whether a problem they are experiencing is due to a tool misusage or, instead, if it is a well-known bug of the tool. This can be done, for example, by capturing characteristics of the error (*e.g.,* the generated stack traces if available) to automatically search on the project's issue tracker and/or on Stack Overflow for related discussions.

D. Process Related

Documentation process issues are discussed in 81 artifacts.

Internationalization (20). This category covers issues related to translation processes, *e.g.,* missing/wrong language translations, the need for reviewing translated documents and rendering problems due to character encoding.

The lack of translated documentation was a recurrent problem (*e.g., "is there a danish translation started?"* [116]), especially when the English documentation was not available, which represented a usage obstacle for several users. This was the case of a project mainly documented in Chinese. In a pull request created to start an English version of the documentation, the main developer apologized for the lack of translation: *"Most user are Chinese, include me. Our English is not good, so sorry"*, to which a user replied *"I could use google translate, but the more effort I have to put into understanding a framework, the less likely it is that I use it"* [117].

Many projects benefited from crowdsourcing the translations, which allowed external users to contribute. To this end, projects that did not have the documentation on code-sharing platforms, discussed whether to move the documentation to obtain more contributions: *"Wonder [...] if we shouldn't push our doc on GitHub to ease contributions"* [118].

Missing guidelines on how to contribute a translation was also a common concern, mostly addressed by providing a page with instructions, *e.g., "we need a webpage describing the basics of how to go about translating the apache docs"* [99].

Character encoding was another typical source of problems in the context of document translation, as developers were often puzzled about the proper encoding to choose. For example, in a mailing list discussion for the simple question *"Which encoding should be used for the .fr files?"* [119], several encodings were suggested, because factors such as file size or encodings better supported by clients were considered.

Traceability (5). This category concerns issues related to the ability to track documentation changes, *i.e.,* to determine where, when, by whom and why a change was performed.

A straightforward solution to keep track of changes in a document was to manage it with a version control system, *e.g., "move wiki to /docs [...] It also means docs are versioned with each new release"* [120]. It is not always feasible, however, to track changes in a version control system. Examples include cases in which the documentation is stored in binary format or in a database. In such cases, preserving traceability by at least versioning some meta-information for each document was a common solution: *"We need a webpage describing [...] and perhaps some standard comments to put at the top of each doc (english version, author, reviewer)"* [99].

Development issues caused by documentation (8). This category covers issues caused by documentation, *i.e.,* unwanted effects of documentation on the development process.

In one interesting case, auto-generated documentation caused issues for the reviewing process of pull requests, as it resulted in noisy diff outputs. As a solution, the developers suggested to split the documentation and the code changes into two separate commits: *"It would be great if we could find a way [...] to defeat the generated files from showing up in the PR diffs, as they overwhelm the diff and make it very hard to review for any other changes. To that end, it would be great if this PR could be squashed into two commits (one with script etc changes, and one with only generated docs"* [108].

Contributing to doc (50). This category covers issues encountered by (internal or external) contributors of documentation while they were reporting/fixing errors or writing new documents. It also includes developers' concerns on supporting external contributors.

We observed that many projects welcome contributions to their documentation from non-members of the projects. To do so, they tried to facilitate the contribution process by offering different aids, as an Apache developer said *"I've tried to lower the barrier [...] to allow anyone to contribute. You can now edit and review change[s] to the jclouds.incubator.apache.org site entirely within your web browser."* [121]. In one scenario, a developer opposed involving a less-known technical solution, namely GitHub pre-commit, into the contribution pipeline and said *"I am reluctant to use precommit hooks to modify the document, as it makes contributions from the community more difficult"* [88]. Indeed, a non-documented and more laborious contribution process can make someone back out of contributing. For instance, a user who wondered how to update an incomplete documentation page stated: *"I don't know where I should modify this page, I have no problem to update it but because I know that cannot be modified directly I don't know where to do it"* [122]. To avoid this situation, well-explained contribution guidelines [123] were provided, sometimes even augmented with a documentation template [89].

Another common issue was related to the lack of knowledge about best practices to write code comments or documentation, *e.g., "how can I document this in JSDoc return type"* [124]. In another example, a user who started a documentation page, sent an email to get feedback on the draft version of the document by adding *"I have just started some user-guide type of documentation [...] Any feedback is welcome"* [125].

Doc-generator configuration (4). This category covers issues related to documentation generators, mostly found in the context of the building process of a project.

An important issue was observed in a thread of the *Apache SystemML* mailing list, where a developer complained about incomplete and outdated API comments due to the project's build configuration that ignores the warnings of the documentation tool [126]. To improve the documentation quality, the developer suggested marking these issues as blockers, with the goal of fixing them in the next release. In another project, developers decided to treat documentation issues warnings as errors, making the build fail: *"So now once warnings are fixed, maybe we could change them into errors, so when somebody makes a mistake it will cause build to fail"* [127].

Discussion and Implications. Many of the issues related to the documentation process concern the way in which external contributors can help in writing, updating and translating documentation. Our findings can be distilled into guidelines for developers to ease the documentation process, which can result in higher-quality documentation and pleased contributors.

⑂ First, developers have to provide contributors with clear guidelines (ideally accompanied by documentation templates) that carefully explain what is expected to be covered in the documentation, how different types of documentation (*e.g.,* code comments) should be written and what the process to contribute is. Second, developers have to consider widely-used code-sharing platforms (*e.g.,* GitHub) to host documentation, where the likelihood of attracting external contributors from all around the world is quite high, which could help with time-consuming tasks such as the translation of documentation. Third, developers have to adopt mechanisms to promote good documentation practices, such as making a build fail when documentation issues are spotted via program analysis (*e.g.,* a new method has been implemented but one of its parameters has not been documented in the Javadoc).

⑂ Another take-away for developers is the need to provide English documentation for their software projects, which would increase their adoption (and, possibly, the contributions).

⚗ Researchers have instead the possibility to work on the optimization of these documentation processes and answer fundamental research questions, such as what constitutes a good contributors guideline (*e.g.,* by surveying software developers). Finally, as already observed in the literature [9], ⚗ advances in the automatic software documentation field are clearly needed. For example, while current static analysis tools used in continuous integration perform simple checks on documentation (*e.g.,* to identify missing comments), the development of approaches that are able to detect more complex *documentation smells* [64] at building time is worthy of investigation.

V. Threats to Validity

Threats to *construct validity* relate to possible measurement imprecision when extracting data used in our study. The automatic mining of developers' documentation discussions based on keywords-matching mechanisms resulted in the retrieval of some false positives (as reported in Table II). These imprecisions were discarded during our manual analysis, thus they did not affect our findings.

In our manual analysis, we based the classification of discussions on what was stated in the analyzed artifacts. It is possible that the information reported in individual artifacts is incomplete, for example due to the fact that an issue was partially discussed in the mailing list and partially via chat.

Threats to *internal validity* concern confounding factors, internal to our study, that can affect the results. They are related to possible subjectiveness introduced during the manual analysis. We mitigated this threat by making sure that each discussion was independently analyzed by two authors and that conflicts were solved by a third author.

Threats to *external validity* represent the ability to generalize the observations in our study. While we analyzed data of different software projects and from diverse data sources, it is possible that our taxonomy of documentation issues depends on the particular set of discussions we analyzed, and that in other contexts developers discuss issues we did not encounter.

VI. Conclusions

We inspected 878 artifacts from four different sources to derive a taxonomy of 162 types of issues faced by developers and users of software documentation. We qualitatively discussed our findings and expose implications for developers and researchers, with the goal of highlighting good practices and interesting research avenues in software documentation.

In essence, our study empirically confirms and complements previous research findings (and common sense): Developers (and users) prefer documentation that is correct, complete, up to date, usable, maintainable, readable and useful.

Given the undeniable value of good documentation, the question is why it is and remains unpopular in software development. We believe that the issues unveiled through our study corroborate, on the one hand, the need for the realization of a vision like the one laid out by Robillard *et al.* [9]: Systems should be capable of documenting themselves automatically. On the other hand, this requires researchers and practitioners to accept the fundamental notion that documentation is not a mere add-on to any software system, but a part of the system itself.

Acknowledgment

We 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), and CHOOSE for sponsoring our trip to the conference.

References

[1] A. Forward and T. C. Lethbridge, "The Relevance of Software Documentation, Tools and Technologies: A Survey," in *Proc. of the 2002 ACM Symp. on Doc. Eng. (DocEng)*. ACM, 2002, pp. 26–33.

[2] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *IEEE Softw.*, vol. 20, no. 6, pp. 35–39, Nov. 2003.

[3] M. Kajko-Mattsson, "A Survey of Documentation Practice within Corrective Maintenance," *Empirical Software Engineering*, vol. 10, no. 1, pp. 31–55, 2005.

[4] J. Zhi, V. Garousi-Yusifoglu, B. Sun, G. Garousi, S. Shahnewaz, and G. Ruhe, "Cost, benefits and quality of software development documentation: A systematic mapping," *Journal of Systems and Software*, vol. 99, pp. 175–198, 2015.

[5] B. Fluri, M. Wursch, and H. C. Gall, "Do code and comments co-evolve? on the relation between source code and comment changes," in *WCRE'07*, Oct 2007, pp. 70–79.

[6] J. C. Chen and S. J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," *Journal of Systems and Software*, vol. 82, no. 6, pp. 981–992, 2009.

[7] M. Linares-Vásquez, B. Li, C. Vendome, and D. Poshyvanyk, "How do developers document database usages in source code? (n)," in *ASE'15*, Nov 2015, pp. 36–41.

[8] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for Java methods," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, Feb 2016.

[9] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong, "On-demand Developer Documentation," in *Proc. of the 33rd IEEE Int. Conf. on Software Maintenance and Evolution (ICSME)*, sep 2017, pp. 479–483.

[10] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the 'hurried' bug report reading process to summarize bug reports," in *Proc. of the 28th IEEE Int. Conf. on Soft. Maintenance (ICSM)*, Sept 2012, pp. 430–439.

[11] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "AUSUM: Approach for unsupervised bug report summarization," in *Proc. of the ACM SIGSOFT 20th Int. Symp. on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 11:1–11:11.

[12] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic Summarization of Bug Reports," *IEEE Trans. on Soft. Eng.*, vol. 40, no. 4, 2014.

[13] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *Proc. of the 17th Working Conf. on Rev. Eng.*, Oct 2010, pp. 35–44.

[14] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for Java methods," in *Proc. of the IEEE/ACM Int. Conf. on Automated Software Engineering*, 2010, pp. 43–52.

[15] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and V. Shanker, "Automatic Generation of Natural Language Summaries for Java Classes," in *21st IEEE Int. Conf. on Program Comprehension (ICPC'13)*. San Francisco, USA: IEEE, 2013, pp. 23–32.

[16] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D'Mello, "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, 2014, pp. 390–401.

[17] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proc. of the 22nd Int. Conf. on Program Comprehension (ICPC 2014)*. ACM, 2014, pp. 279–290.

[18] P. W. McBurney, C. Liu, C. McMillan, and T. Weninger, "Improving topic model source code summarization," in *Proc. of the 22nd Int. Conf. on Program Comprehension (ICPC 2014)*. ACM, 2014, pp. 291–294.

[19] M. Linares-Vásquez, B. Li, C. Vendome, and D. Poshyvanyk, "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, 2016, pp. 270–281.

[20] S. Panichella, A. Panichella, M. Beller, A. Zaidman, and H. C. Gall, "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, 2016, pp. 547–558.

[21] B. Li, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and N. A. Kraft, "Automatically documenting unit test cases," in *Proc. of the IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2016)*, April 2016, pp. 341–352.

[22] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "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*, Sept 2014, pp. 275–284.

[23] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, "ChangeScribe: A tool for automatically generating commit messages," in *2015 IEEE/ACM 37th IEEE Int. Conf. on Software Engineering*, vol. 2, May 2015, pp. 709–712.

[24] S. Jiang and C. McMillan, "Towards automatic generation of short summaries of commits," in *Proc. of the 25th IEEE/ACM Int. Conf. on Program Comprehension (ICPC 2017)*, May 2017, pp. 320–323.

[25] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, "Automatic generation of release notes," in *Proc. of the 22nd ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (FSE 2014)*. ACM, 2014, pp. 484–495.

[26] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, A. Marcus, and G. Canfora, "ARENA: An approach for the automated generation of release notes," *IEEE Trans. on Soft. Eng.*, vol. 43, no. 2, Feb 2017.

[27] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "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, 2016, pp. 499–510.

[28] A. T. T. Ying and M. P. Robillard, "Code fragment summarization," in *Proc. of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013)*. ACM, 2013, pp. 655–658.

[29] R. Krasniqi, S. Jiang, and C. McMillan, "Tracelab components for generating extractive summaries of user stories," in *2017 IEEE Int. Conf. on Soft. Maint. and Evolution (ICSME)*, Sept 2017, pp. 658–658.

[30] R. Holmes and G. C. Murphy, "Using structural context to recommend source code examples," in *Proc. of the 27th Int. Conf. on Software Engineering (ICSE 2005)*. ACM, 2005, pp. 117–125.

[31] J. Stylos and B. A. Myers, "Mica: A web-search tool for finding api components and examples," in *Proc. of the Visual Languages and Human-Centric Computing (VLHCC 2006)*. IEEE, 2006, pp. 195–202.

[32] G. C. Murphy, R. J. Walker, and R. Holmes, "Approximate structural context matching: An approach to recommend relevant examples," *IEEE Transactions on Software Engineering*, vol. 32, pp. 952–970, 12 2006.

[33] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, and A. Marcus, "How can I use this method?" in *Proc. of the 37th IEEE/ACM Int. Conf. on Software Engineering (ICSE 2015)*, 2015, pp. 880–890.

[34] S. Thummalapenta and T. Xie, "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, 2007, pp. 204–213.

[35] S. P. Reiss, "Semantics-based code search," in *Proc. of the 31st Int. Conf. on Soft. Eng. (ICSE 2009)*. IEEE, 2009, pp. 243–253.

[36] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: Finding relevant functions and their usage," in *Proc. of the 33rd Int. Conf. on Soft. Eng. (ICSE 2011)*. ACM, 2011, pp. 111–120.

[37] L. Ponzanelli, S. Scalabrino, G. Bavota, A. Mocci, R. Oliveto, M. D. Penta, and M. Lanza, "Supporting software developers with a holistic recommender system," in *Proc. of the 39th IEEE/ACM Int. Conf. on Software Engineering (ICSE)*, May 2017, pp. 94–105.

[38] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Leveraging crowd knowledge for software comprehension and development," in *Proc. of the 17th European Conf. on Soft. Maint. and Reeng.*, March 2013, pp. 57–66.

[39] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "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, 2014, pp. 102–111.

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

[41] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation," *Proc. of the 18th ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (FSE 2010)*, p. 127, 2010.

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

[43] R. Plösch, A. Dautovic, and M. Saft, "The Value of Software Documentation Quality," in *Proc. of the 14th Int. Conf. on Quality Software*, oct 2014, pp. 333–342.

[44] G. Garousi, V. Garousi-Yusifoglu, G. Ruhe, J. Zhi, M. Moussavi, and B. Smith, "Usage and usefulness of technical software documentation: An industrial case study," *Information and Software Technology*, vol. 57, no. 1, pp. 664–682, 2015.

[45] G. Uddin and M. P. Robillard, "How API Documentation Fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, 2015.

[46] N. Alhindawi, O. M. Al-Hazaimeh, R. Malkawi, and J. Alsakran, "A Topic Modeling Based Solution for Confirming Software Documentation Quality," *Int. Journal of Advanced Computer Science and Applications*, vol. 7, no. 2, pp. 200–206, 2016.

[47] S. M. Sohan, F. Maurer, C. Anslow, and M. P. Robillard, "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*, vol. 2017-October, pp. 53–61, 2017.

[48] J. D. Arthur and K. T. Stevens, "Document quality indicators: A framework for assessing documentation adequacy," *Journal of Software Maintenance: Research and Practice*, vol. 4, no. 3, pp. 129–142, 1992.

[49] A. Dautovic, "Automatic assessment of software documentation quality," in *2011 26th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE 2011)*, nov 2011, pp. 665–669.

[50] T. C. Lethbridge, J. Singer, and A. Forward, "Use Documentation : The State of the Practice Documentation," *Ieee Focus*, p. 5, 2003.

[51] M. Visconti and C. R. Cook, "Assessing the State of Software Documentation Practices," in *Product Focused Software Process Improvement*, 2004, pp. 485–496.

[52] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk, "An exploratory analysis of mobile development issues using Stack Overflow," in *Proc. of the 10th Working Conf. on Mining Software Repositories (MSR)*, May 2013, pp. 93–96.

[53] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in Stack Overflow," *Empirical Softw. Eng.*, vol. 19, no. 3, pp. 619–654, Jun. 2014.

[54] C. Rosen and E. Shihab, "What are mobile developers asking about? A large scale study using Stack Overflow," *Empirical Softw. Eng.*, vol. 21, no. 3, pp. 1192–1223, Jun. 2016.

[55] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about?" *IEEE Software*, vol. 32, no. 3, pp. 70–77, May 2015.

[56] "Replication Package," https://github.com/REVEAL-ICSE19-DocIssues/ReplicationPackage.

[57] I. Grigorik, "GitHub Archive," https://www.githubarchive.org.

[58] GitHub, "Event Types & Payloads," https://developer.github.com/v3/activity/events/types/.

[59] A. S. Foundation, "Apache Mail Archives," http://mail-archives.apache.org/mod_mbox/.

[60] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. on Soft. Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[61] R. Koschke, "Survey of research on software clones," in *Duplication, Redundancy, and Similarity in Software*, ser. Dagstuhl Seminar Proceedings, no. 06301. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2007/962

[62] S. Scalabrino, M. Linares-Vásquez, R. Oliveto, and D. Poshyvanyk, "A comprehensive model for code readability," *Journal of Software: Evolution and Process*, vol. 30, no. 6, 2018.

[63] J. Kincaid, R. Fishburne, R. Rogers, and B. Chissom, "Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel," Institute for Simulation and Training, Tech. Rep., 1975.

[64] H. Zhong and Z. Su, "Detecting API documentation errors," *SIGPLAN Not.*, vol. 48, no. 10, pp. 803–816, Oct. 2013.

Artifacts' References

[65] "GitHub Issue of acid-state/acid-state." http://bit.ly/2wokNDJ

[66] "GitHub Issue of rockstor/rockstor-core." http://bit.ly/2wr2AFo

[67] "GitHub Issue of pluskid/mocha.jl." http://bit.ly/2wop8H1

[68] "GitHub PR of silverstripe/silverstripe-framework." http://bit.ly/2wnCvXQ

[69] "Apache Mailing List httpd-docs." http://bit.ly/2wqx4Y2

[70] "Apache Mailing List httpd-docs." http://bit.ly/2wmverj

[71] "GitHub Issue of tinymce/tinymce." http://bit.ly/2wmTIAB

[72] "GitHub Issue of mlpack/mlpack." http://bit.ly/2wsvBAS

[73] "GitHub Issue of elliotchance/c2go." http://bit.ly/2wmSk0T

[74] "GitHub PR of falconry/falcon." http://bit.ly/2wixz6s

[75] "GitHub Issue of trilinos/trilinos." http://bit.ly/2wmAM57

[76] "GitHub Issue of bytedeco/javacpp-presets." http://bit.ly/2wnfqEF

[77] "GitHub Issue of nodemcu/nodemcu-firmware." http://bit.ly/2wqzMgt

[78] "GitHub Issue of facebook/watchman." http://bit.ly/2wpzpmb

[79] "Apache Mailing List forrest-dev." http://bit.ly/2wr91Z6

[80] "GitHub PR of coreos/etcd." http://bit.ly/2wr24aq

[81] "GitHub Issue of d3/d3-dispatch." http://bit.ly/2wmBQWE

[82] "GitHub PR of alibaba/rax." http://bit.ly/2wmk4TC

[83] "Apache Mailing List systemml-dev." http://bit.ly/2wqU6y2

[84] "GitHub Issue of domaindrivendev/swashbuckle." http://bit.ly/2wo2oa6

[85] "GitHub Issue of doctrine/doctrine1." http://bit.ly/2wodZWN

[86] "GitHub Issue of webpack/docs." http://bit.ly/2wmW56t

[87] "GitHub Issue of stevegrunwell/mcavoy." http://bit.ly/2wp3QZB

[88] "GitHub PR of asciidoctor/asciidoctorj." http://bit.ly/2woguZb

[89] "Apache Mailing List cocoon-docs." http://bit.ly/2BJ8g3x

[90] "Apache Mailing List directory-dev." http://bit.ly/2wvX1G3

[91] "GitHub Issue of revapi/revapi." http://bit.ly/2wlugeZ

[92] "GitHub PR of habitat-sh/habitat." http://bit.ly/2wogWql

[93] "GitHub Issue of riot-os/riot." http://bit.ly/2wssVTQ

[94] "StackOverflow discussion 30596247." http://bit.ly/2wrbAdG

[95] "GitHub Issue of realm/jazzy." http://bit.ly/2wr2lKu

[96] "Apache Mailing List httpd-docs." http://bit.ly/2wjbN2C

[97] "Apache Mailing List stdcxx-user." http://bit.ly/2wmJDDN

[98] "Apache Mailing List hc-dev." http://bit.ly/2woTcCj

[99] "Apache Mailing List httpd-docs." http://bit.ly/2wlrsP5

[100] "GitHub PR of rails/rails." http://bit.ly/2wmRKQL

[101] "GitHub PR of paulcollett/vue-masonry-css." http://bit.ly/2wr28qG

[102] "GitHub PR of composewell/streamly." http://bit.ly/2woqG3N

[103] "GitHub PR of facebook/react-native." http://bit.ly/2wstqgG

[104] "GitHub Issue of commercialhaskell/stack." http://bit.ly/2wkjxBu

[105] "StackOverflow discussion 532779." http://bit.ly/2wnXfii

[106] "StackOverflow discussion 1136234." http://bit.ly/2wp0wO7

[107] "StackOverflow discussion 48435375." http://bit.ly/2wnz6bw

[108] "GitHub PR of uber/luma.gl." http://bit.ly/2wooeKD

[109] "StackOverflow discussion 23900027." http://bit.ly/2woWIwv

[110] "GitHub Issue of pinax/pinax-badges." http://bit.ly/2wls0o7

[111] "GitHub PR of netflix/hollow." http://bit.ly/2wqAcU5

[112] "Apache Mailing List camel-dev." http://bit.ly/2wvWQun

[113] "StackOverflow discussion 45737685." http://bit.ly/2wjeH7w

[114] "GitHub Issue of phpdocumentor/phpdocumentor2." http://bit.ly/2wlsTwR

[115] "StackOverflow discussion 23689297." http://bit.ly/2MzKoVi

[116] "Apache Mailing List httpd-docs." http://bit.ly/2wr9GK4

[117] "GitHub Issue of dvajs/dva." http://bit.ly/2wotluk

[118] "Apache Mailing List tomee-dev." http://bit.ly/2woTzwH

[119] "Apache Mailing List httpd-docs." http://bit.ly/2wpwc67

[120] "GitHub PR of keratin/authn." http://bit.ly/2wnBb7k

[121] "Apache Mailing List jclouds-user." http://bit.ly/2wmxJtG

[122] "Apache Mailing List tomee-users." http://bit.ly/2MJkxJJ

[123] "Apache Mailing List jena-dev." http://bit.ly/2wquDoF

[124] "StackOverflow discussion 45342178." http://bit.ly/2wluSBp

[125] "Apache Mailing List tuscany-user." http://bit.ly/2wqudyB

[126] "Apache Mailing List systemml-dev." http://bit.ly/2o7WupT

[127] "GitHub PR of prestodb/tempto." http://bit.ly/2wp1ZnB