Challenges and Solutions of Free and Open Source Software Documentation: A Systematic Mapping Study

Giniele Pinho
Federal University of Ceará
Crateús, CE, Brazil
giniele@alu.ufc.br

Aguiar Jeová Caçula
Federal University of Ceará
Crateús, CE, Brazil
jeova.junior@alu.ufc.br

Lucas Costa
Federal University of Cariri
Juazeiro do Norte, CE, Brazil
lucas.germano@aluno.ufca.edu.br

Igor Wiese
Federal University of Technology - Paraná
Campo Mourão, Paraná, Brazil
igor@utfpr.edu.br

Allysson Allex Araújo
Federal University of Cariri
Juazeiro do Norte, CE, Brazil
allysson.araujo@ufca.edu.br

ABSTRACT

Software documentation is a relevant process for delivering quality software, as it assists stakeholders in using, understanding, maintaining, and implementing software productively. However, notable particularities emerge when investigating the context of Free and Open Source Software (FOSS) projects, which require special attention. Therefore, through a Systematic Mapping Study (SMS), this work aims to map the challenges and solutions regarding software documentation in FOSS based on the last ten years of scientific research (published between 2013 and 2023). From an initial set of 1271 papers, 12 primary studies were identified from which it was possible to categorize five challenges (Collaboration, Quality, Incompleteness, Maintainability, and Categorization) and three general perspectives of solutions (Strategic Use of README, Adoption of Artificial Intelligence, and Support Tools & Approaches). As an academic contribution, we provide an SMS revealing a set of challenges and solutions related to software documentation, a topic still underexplored in the FOSS research context. From a practical and industrial standpoint, this paper promotes a reflection on the use of documentation in FOSS projects, echoing challenges and solutions that can contribute to improving the quality of documentation.

CCS CONCEPTS

• **Software and its engineering** → **Documentation**; **Open source model**.

KEYWORDS

Software Documentation, Free Software, Open Source, Systematic Mapping Study

1 INTRODUCTION

Software documentation plays a fundamental role in the development of high-quality software [52]. This importance stems from documents' ability to assist stakeholders in using, understanding, maintaining, and developing a system more productively [2, 26]. As discussed by Parnas [45] and Forward [22], software documentation is an artifact designed to convey information about the related software system, serving as a written description that can be used as evidence for knowledge management. High-quality documentation is one of the critical factors in producing high-quality software [10]. Hence, the documentation is expected to provide accurate and valuable information about systems, both in closed-source and open-source projects.

Free and Open Source Software (FOSS) are solutions in which users can freely distribute, access, adapt, modify, and redistribute the source code for their use and the benefit of the community [36]. The community of a FOSS project consists of contributors (developers, users, etc.) who share a common interest in collaborating on software solutions [34, 62]. Adopting a FOSS model has proven to be a strategic opportunity for organizations in various domains, as it offers benefits such as access to talented programmers worldwide, potential for peer review, and the ability to freely distribute, adapt, and modify the source code [23]. This strategic perspective emphasizes the potential for increased innovation and software quality among competitive organizations [66].

Despite the well-known advantages of the FOSS model, such as the flexibility of having contributors from anywhere and the substantial potential for peer review, it also faces distinctive particularities [34, 42]. For instance, there is the voluntary nature of participation, which can lead to fluctuating levels of contributors engagement, the lack of centralized management to direct developer resources and attributes, the diversity in skills and experience among participants, and the need to recruit and retain new contributors [18, 38]. In light of these characteristics, software documentation, as a process, is impacted and consequently requires special attention, given that it promotes valuable benefits for software development, such as supporting asynchronous communication and fostering knowledge management [42]. As stated by Aghajani et al. [3], developers (and users) prefer documentation that is correct, complete, up to date, usable, maintainable, readable and useful.

While software documentation is a widely investigated research topic in the Software Engineering (SE) community [2, 17, 24, 26, 53, 69], there is a research gap concerning the systematic mapping of challenges and solutions in software documentation for FOSS. Aiming to understand the recent state-of-the-art in this intersection of studies involving software documentation and FOSS, this work proposes to investigate the following research question: *In the past ten years, what are the challenges and solutions regarding software documentation in FOSS*? Specifically, the choice of a ten-year timeframe is strategic and justified, as it allows us to focus on recent perspectives that reflect the current landscape of SE in FOSS. This

timeframe also ensures that we capture the up-to-date challenges and solutions, considering the rapid evolution and changes in the FOSS over the past decade. In addition, this period includes the rise of key technologies and methodologies that have impacted FOSS development, such as the widespread adoption of GitHub and the maturation of DevOps practices.

Considering the research question's scope, this work adopts a methodological approach based on a Systematic Mapping Study (SMS). In particular, the SMS is deemed appropriate as its objective is to review, classify, and structure papers related to a specific research field [48]. In this sense, the use of SMS is justified by its focus on delineating a specific research field, providing an overview of the topic, and enabling the categorization of the research topic of interest [49]. From an initial set of 1271 papers, 12 primary studies were identified, from which it was possible to categorize five major challenges (Collaboration, Quality, Incompleteness, Maintainability, and Categorization) and three general perspectives of investigated solutions (Strategic use of README, Adoption of Artificial Intelligence, and Support Tools & Approaches).

This study delivers contributions to academia and practice in the context of SE by: 1) Unveiling a set of challenges and solutions related to software documentation, a topic relatively underexplored in FOSS; 2) Providing an overview of the literature covering software documentation and FOSS, thereby consolidating knowledge and identifying trends and gaps; and 3) Promoting a concrete reflection on the documentation process in FOSS projects, echoing practical challenges and solutions that can support improving the quality of documentation and the efficiency of such projects.

This paper is organized as follows. Section 2 presents the theoretical foundation and related works. Section 3 clarifies our research method, including the protocol for the SMS. Section 4 discusses the results and analyses based on the primary studies, while Section 5 presents a general discussion of our findings. Section 6 addresses threats to validity of this study. Finally, Section 7 approaches our concluding remarks and perspectives of future work.

2 BACKGROUND

This section articulates the literature review and related works to provide an overview of the literature and how our work is positioned within it in terms of contributions. Firstly, we discuss the context of Free and Open Source Software. Then, we overview the scientific literature covering software documentation.

2.1 Free and Open Source Software

Free and Open Source Software (FOSS) projects are quite relevant in the modern software industry. Currently, many software systems relevant to society rely on FOSS projects [7]. According to Van Angeren et al. [63], the overarching concept behind free software encompasses software artifacts, including source code, licenses, development best practices, innovation, ethics, philosophy, social movement, community, culture, governance, and organizational engagement. Then, software emerges from a coordinated and unsupervised community of developers and other contributors. This development process is based on the principle that software should be freely shared among users, granting them the ability to introduce implementations and modifications [30]. Differing from traditional and closed source software development, FOSS projects encompass free redistribution, availability of source code, and the possibility of modifications, which must be distributed under the same terms as the original software [47].

As Von Krogh and Von Hippel [64] note, FOSS is commonly created by an individual or a group of people aiming to develop a software product to meet their needs. The initial version of the created source code is made freely available to everyone, allowing anyone to participate in the software development cycle [68]. Stamelos et al. [58] emphasize two principles broadly defining the power of FOSS: rapid evolution so that many users/programmers can use the new system and modify it, with no time spent on 'unnecessary' management activities and many programmers working simultaneously on the same problem.

The FOSS community comprises contributors who share a common interest and interact to share knowledge for project development [34]. Individuals join such communities at different ages and with diverse backgrounds, abilities, resources, and objectives, along with varying levels of programming skills and experience [38]. According to Steinmacher et al. [59], the successful development of a FOSS project essentially relies on the work of a community of volunteer developers distributed globally and collaborating via the Internet. In addition, Michlmayr et al. [42] highlight advantages of the FOSS development model, such as the potential for peer review and attracting excellent programmers worldwide.

However, FOSS projects can be complex to manage and involve continuous improvement processes. According to Jalote [33], the software process comprises a set of activities that need to be interconnected by standards, and if the activities operate correctly according to these standards, the desired outcome is achieved. Hence, FOSS communities are examples of shared knowledge involving assets from different individuals in the process of interaction and learning [31]. Active community involvement is essential for the success of a FOSS project, requiring efforts to increase community participation and engagement through the use of tools, practices, and processes [34]. Michlmayr et al. [42] explain three development and quality practices essential for a good FOSS project development structure, which are discussed below.

FOSS projects heavily depend on **Infrastructure**, which enables distributed development and collaboration. Important parts of the infrastructure include Bug Tracking Systems to gather feedback from users and reports of actual bugs as well as feature requests; Version Control Systems to allow multiple people to work on the same code simultaneously and track who makes which changes; Automated Builds to ensure that the latest code in the version control system is still compiling and Mailing Lists for communication among developers and users.

Regarding the **Process**, FOSS development involves different tasks, but some are not necessarily documented, and developers adhere to them implicitly. Possible processes include: i) Onboarding: projects require potential members to follow specific, mostly undocumented procedures to join a project. These procedures vary considerably among projects. Some explicitly ensure admitting only contributors from whom high-quality submissions can be expected, while other projects are more flexible; ii) Peer review: changes made in the version control system are typically reviewed by project members, although this form of peer review is often not well formalized;

iii) Testing: to ensure that a new version meets the standards of a project and has no major regressions, some projects have testing checklists. These checklists contain the most critical functions and briefly describe how they can be tested and iv) Quality assurance: some projects organize bug days or bug-solving parties to triage their pending bugs. During this work, duplicate bug reports are marked as such, old bugs are reproduced, and bugs are also fixed.

Finally, **Documentation** in FOSS projects regarding development practices should be explicit, including ways to contribute and how to join the project. There are two types of documentation that are commonly used. The first type is coding style documentation, which is intended for developers and outlines the style to be used for source code. The second type is code commit documentation, which specifies who can make changes to a project's version control system and when those changes can be made.

FOSS development often rely on self-assignment of tasks with little coordination, but all developers need to align their activities toward a shared goal [43]. Consequently, coordinating and organizing work in FOSS projects usually involves balancing effort demand (desired resources and known bugs that will take time and specialized skills to fix) with effort supply (volunteers and paid developers who have their motivations and priorities) [36]. In this context, software documentation stands out as a valuable guide providing knowledge to stakeholders, aiding in understanding a particular system, and contributing to task efficiency [27].

2.2 Software Documentation

Software documentation can be defined as an process whose purpose is to communicate information about the software system to which it belongs [22]. In other words, documentation establishes communication among all involved parties and levels their knowledge about the project, aiming to transfer and share knowledge [57]. Software documentation is considered an integral part of the software development process and is therefore utilized in various ways throughout the software lifecycle, such as being a means of communication among stakeholders, serving as a repository of information for maintaining up-to-date software or acting as a guide for onboarding new developers [18]. Thus, documentation comprises a set of general and technical manuals, organized in the form of texts and comments, utilizing tools like dictionaries, diagrams and flowcharts, graphs, drawings, among others [15].

As Chapin [12] highlights, software can be determined by many factors, among which documentation, including its preparation and maintenance, can be emphasized. Consequently, attention must be paid to the quality, obsolescence, or lack of content in documentation. However, creating qualified documentation is not trivial, and the diversity of existing document models also raises questions [57]. For example, Coelho [15] emphasizes that developers should find tools that facilitate understanding of documentation, both for administrators and future software users. Moreover, Parnas [45] reinforces that software documentation needs to be correct and precise, meaning that the information one can obtain from the document should be true for the system being described, leaving no doubts about what they mean.

According to Michelazzo [41], software documentation in the software development cycle can be organized into two major groups. The first group concerns to **Technical Documentation**, considered more straightforward as it describes the developer's work. Geared towards developer use, this documentation group comprises dictionaries and data models, process flowcharts and business rules, function dictionaries, and code comments. The second group refers to **Usage Documentation** and is considered more complex, requiring special skills for writing manuals, screenshots, drawings, and other graphical elements. Thus, this group may focus on the end-user and system administrator, consisting of handbooks or manuals that present how the system should be used, what to expect from it, and how to obtain the desired information [2].

Garousi [25] further emphasizes that when a document is created, it can be used as a tutorial report or as a reference document. According to Cioch et al. [13], there are four stages of experience in a software project (from newcomer, the first day at work; to expert, after a few years of work on a system). For each stage, different documents are proposed: newcomers need a brief overview of the system; learners need the system architecture; interns need task-oriented documents, such as requirements description, process description, examples, step-by-step instructions; finally, experts need low-level documentation, as well as requirements and design descriptions. In the view of Ambler [5], there are two basic reasons for documenting software: to assist communication during software development and to aid understanding in maintenance and update activities when necessary.

Michelazzo [41] describes four common types of software documentation. First, there is **Code Documentation**, which is typically achieved through comments within the code itself and the generation of online documentation. Developers should be aware that they will not be the sole individuals working on the system and therefore must comment their code clearly. Next, **Data Models** graphically and logically represent a system's database, including relationships, entities, keys, and all related data aspects. They are foundational for system development and are usually created before development starts, either through reverse engineering or based on application needs. Then, there are **Data Dictionaries**, files or documents defining the basic organization of a database's data, including tables, fields, definitions, types, and descriptions. Finally, **Flowcharts** visually present the logical sequence of information in a process or system, using geometric elements to indicate different process parts. They offer an overview of the system's logic, from high-level processing to minor components, aiding in understanding what needs to be done within the system.

Ambler [5] also stress that software documentation meets three needs: (i) contractual; (ii) supporting a software development project by allowing team members to conceive the solution to be implemented gradually and (iii) enabling a software development team to communicate implementation details over time to the maintenance team. Therefore, software documentation is fundamental for FOSS projects as it addresses two basic principles: collaborative production and wide dissemination. When more than one person is developing a work, communication becomes paramount [29].

As projects with open and collaborative content, FOSS communities strive to keep documentation updated. In this sense, FOSS projects have intrinsic organization characteristics, mainly because they must deal with developer communities [51]. Winters et al. [67],

for example, discuss that onboarding new team members or codebase requires much less effort if the process is clearly documented. Hence, contributors are more focused when project design goals are clearly stated. In summary, software documentation proves to be an invaluable resource for any software project, as it helps stakeholders use, understand, maintain, and develop a system [2].

During our literature analysis, we also found secondary studies that were concerned with software documentation; however, none explicitly focused on the context of challenges and solutions faced by FOSS. Zhi et al. [69] employed a systematic-mapping methodology to map the existing knowledge concerning software documentation cost, benefit, and quality. They found that the documentation cost aspect appears to have been neglected in the existing literature, and there are no systematic methods or models to measure cost-effectively. Additionally, Theunissen et al. [61] conducted a SMS to identify and analyze research on documentation in Continuous Software Development. They observed challenges such as understanding informal documentation, the perception of documentation as waste, productivity measurement based solely on working software, inconsistencies between documentation and software, and a short-term focus. Furthermore, they identified practices related to non-written and informal communication, using development artifacts for documentation, and adopting architecture frameworks.

Habib and Romli [28] conducted a SMS focused on exploring the issues and importance of documentation in agile development. From this study, the authors identified 14 aspects related to documentation in agile that were investigated, highlighting the importance of having "just enough" documentation in agile methodologies. Still in the context of agile software development, Islam et al. [32] conducted a Systematic Literature Review (SLR) aiming to systematically identify what to document, which documentation tools and methods are in use, and how those tools can overcome documentation challenges. Ding et al. [19] also employed an SLR method to identify primary studies on knowledge-based approaches in software documentation. They observed that architecture understanding is the primary benefit of using knowledge-based approaches in software documentation. However, they also noted that the cost of retrieving information from documents remains a major concern when employing knowledge-based approaches in documentation.

3 RESEARCH METHOD

The methodological path taken in this study was grounded in a Systematic Mapping Study (SMS), aiming to map, based on the last decade of scientific research, the challenges and solutions regarding software documentation in FOSS. Unlike a Systematic Literature Review (SLR) that involves a rigorous critical evaluation of selected studies, SMS is more exploratory and focuses on mapping existing studies to provide an overview of the research field under investigation [35].

Figure 1 presents an overview of our methodological plan comprising three major stages. In **Stage 1 (Planning)**, exploratory searches were manually conducted using Google Scholar to refine criteria, research questions, synonyms for the search string, and information extraction model. Our research protocol was defined drawing inspiration from the SMS by Trinkenreich [62], while adhering to the well-known guidelines specified by Petersen et al. [48] and Kitchenham et al. [35].

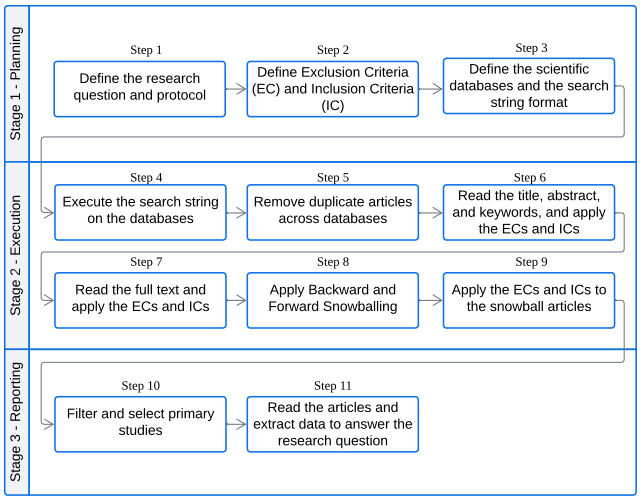

Figure 1: Methodological procedures

With the aim of selecting works that explicitly address the topic of software documentation in FOSS, different Inclusion Criteria (IC) and Exclusion Criteria (EC) were defined for paper filtering. If a paper met an EC, it was promptly removed. Additionally, a paper could only be retained if it met all ICs. Thus, the ICs and ECs defined for the screening of works were as follows:

- **(+) IC1:** Papers that contain the search expression's keywords in the abstract, title, and/or article keywords of the selected paper;
- **(+) IC2:** Papers published in peer-reviewed venues, including workshops, conferences, symposiums, and journals;
- **(+) IC3:** Papers written in English;
- **(-) EC1:** Papers not available in full text;
- **(-) EC2:** Papers that lack an abstract;
- **(-) EC3:** Papers where software documentation is used only as a superficial example;
- **(-) EC4:** Secondary or tertiary papers (*e.g.*, systematic literature reviews, literature surveys, etc.);
- **(-) EC5:** Papers in the form of tutorials, editorials, etc., as they do not provide sufficient information;
- **(-) EC6:** Papers that do not meet the inclusion criteria.

Moreover, the selected databases were IEEE Xplore, Web of Science, and Scopus, because they (i) index relevant sites for this study, (ii) support searches using a Boolean expression, (iii) provide access to full texts, and (iv) allow the export of properly formatted results. The search string was defined with terms relevant to the research objective: "(`open source` OR `open-source` OR `OSS` OR `free software` OR `FOSS OR FLOSS`) **AND** (`software development` OR `software engineering`) **AND** (`documentation` OR `document` OR `guide` OR `writing` OR `artifact` OR `specification` OR `notes` OR `knowledge base` OR `report` OR `whitepaper`)". We conducted exploratory searches using Google Scholar to establish the search

terms. From our exploratory search and experience, we also chose three relevant studies as control papers [8, 11, 54]. In our pilot studies to test the protocol, we found all three control papers. As can be seen, we seek to combine various terms and synonyms related to FOSS development and software documentation. In this case, we covered different variations such as `open source` and `open-source`, as well as acronyms like `OSS` and `FOSS`. We also included terms related to documentation, such as `guide`, `writing`, `artifact`, `specification`, `notes`, `knowledge base`, `report`, and finally, `whitepaper`. Works explicitly addressing documentation in FOSS projects with an empirical research method were considered. The search was conducted in July and August 2023.

In **Stage 2 (Execution)**, the search string was executed in the selected databases, returning 574 papers in IEEE Xplore, 457 papers in Web of Science, and 240 papers in Scopus, totaling initially 1271 papers. The filtered papers were organized into a spreadsheet in Google Sheets for better visualization and data organization. Firstly, duplicate papers between the databases were removed. Thus, 164 duplicate papers were removed, leaving 1107 papers to be analyzed. The filtered papers were then divided into two groups to leverage the reliability and rigor, with two co-authors applying the ECs and ICs (one person per group), while the main author performed a double-check on all papers (from both groups). The researchers met virtually to resolve decision conflicts and reached a preliminary selection consensus. Two more experienced authors were invited to discuss the final decision in case conflicts persisted. Thus, in addition to the main author, four other individuals were involved in the paper filtering process, reinforcing the rigor of the explored research protocol.

During paper filtering, researchers used a dialectical approach for decision-making where each paper argued for keeping or removing the paper, and then a consensus was reached after debates [56]. This approach helped systematically evaluate arguments for keeping or removing a paper and avoid decision-making bias by heavily relying on initial impressions about a paper. After applying the ECs and ICs, the set was reduced to 11 papers. From the list of 11 papers, a backward snowballing approach was independently applied to complement the obtained papers. In this sense, we manually reviewed all the bibliographic references of each primary study. Duplicates and references not within the ten-year period stipulated by the research criteria were excluded. Subsequently, we applied the inclusion and exclusion criteria to each reference. As a result of the backward snowballing, one more paper was included in the primary studies, totaling a final list of 12 papers. We also conducted the Snowball Forward process looking to the citations of our primary studies, but no new primary study was found in line with the inclusion and exclusion criteria. Figure 2 synthesizes our step-by-step selection process.

Finally, in **Stage 3 (Reporting)**, the final list of papers was organized for the purpose of generating the analytical report. A total of 12 primary studies were selected. Concerning the data extraction, the studies were organized in a spreadsheet in Google Sheets with standard data fields: Title of the work, authors' names, country, keywords, year of publication, type of publication, publication venue, database, identified challenges, and explored solutions. Table 1 summarizes the 12 papers that form the primary sources for the study. All research data is available in the article's support repository [50], thus promoting transparency and accessibility to the processes explored in our SMS.

A qualitative-quantitative perspective was explored for the data analysis. Regarding quantitative analysis, descriptive statistics were used to analyze data on distribution by year of publication, studies by databases, and publication venues. As for the qualitative perspective, open coding was explored with the aim of categorizing and obtaining relevant data from the studies to answer our research question [55]. Initially, the first author conducted the coding independently, deriving them from the excerpts in the papers. These post-formed codes were then associated and categorized following a chain of evidence. To ensure reliability, two other experienced co-authors with expertise in Software Engineering research reviewed and discussed the codes until reaching consensus. Four meetings were held among these authors to refine the codes and increase the reliability of the results. All steps of the study, including the extracted codes, are documented in our Zenodo package [50].

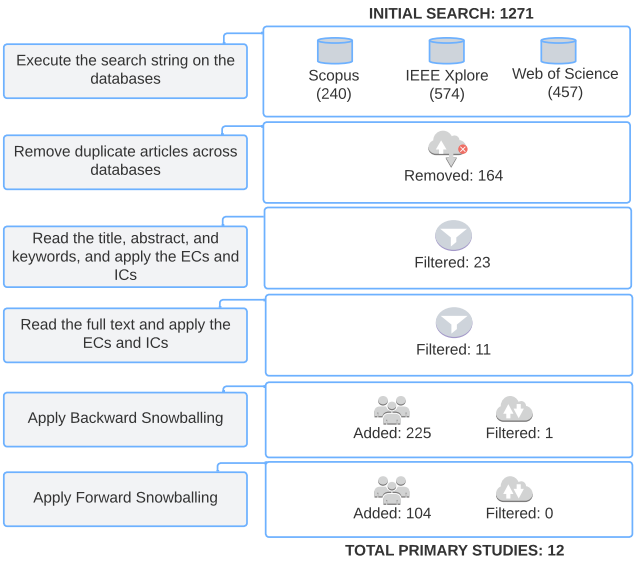

Figure 2: Step by step selection of primary studies

4 RESULTS AND ANALYSIS

4.1 Primary Studies Characterization

Figure 3 displays the distribution of primary studies by year. The filtering period spanned from 2013 to 2023. There is a four-year interval without publications, from 2013 to 2023. The oldest papers were published in 2014. Since 2021, at least one article has been published each year. 2014 and 2023 had the highest number of published papers, with three papers each.

Regarding the databases and publication venues, it was found that Web of Science had the highest number of selected papers, accounting for 41.7% (5 papers), followed by IEEE Xplore with 33.3% (4 papers), and Scopus with 25% (3 papers). As for publication venues, three of the twelve selected primary studies were published in the International Conference on Mining Software Repositories

Table 1: List of Primary Studies (PS).

ID	References	Year	Publication Venue
PS01	Carvalho et al. [11]	2014	Computer Science and Information Systems
PS02	Ding et al. [20]	2014	International Conference on Engineering of Complex Computer Systems
PS03	Bigliardi et al. [9]	2014	International Conference on Quality Software
PS04	Aversano et al. [8]	2017	International Conference on Evaluation of Novel Approaches to Software Engineering
PS05	Ma et al. [40]	2018	International Conference on Mining Software Repositories
PS06	Prana et al. [53]	2019	Empirical Software Engineering
PS07	AlOmar et al. [4]	2021	Journal of Software: Evolution and Process
PS08	Pasuksmit et al. [46]	2022	International Conference on Mining Software Repositories
PS09	Puhlfürß et al. [54]	2022	International Conference on Software Maintenance and Evolution
PS10	Sun et al. [60]	2023	International Conference on Mining Software Repositories
PS11	Wermke et al. [65]	2023	Symposium on Security and Privacy
PS12	Ciurumelea et al. [14]	2023	Empirical Software Engineering

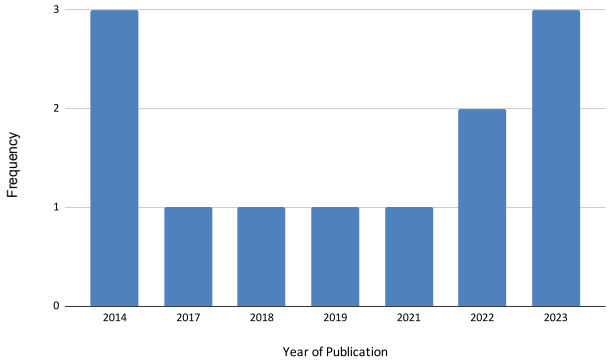

Figure 3: Year of publication of primary studies

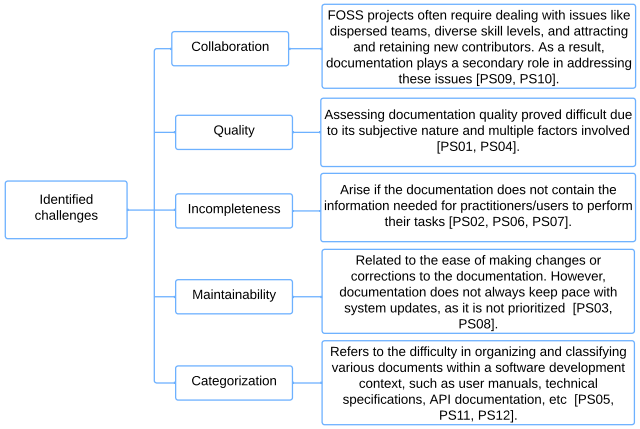

Figure 4: Summary of the identified challenges

(MSR). The remaining papers were distributed among different areas and events/journals. Additionally, the results showed that seven papers (58.3%) were published in Conferences. The journal with the most articles (2) was Empirical Software Engineering (EMSE).

4.2 Challenges

The literature presents different perspectives regarding the use of documentation in FOSS projects, leading to different challenges faced by stakeholders. In theory, good documentation is an invaluable resource for any software project, as it helps stakeholders use, understand, maintain, and develop a system [2]. Therefore, recognizing that documentation assists software engineers in performing maintenance and development tasks more efficiently, it is necessary to understand the recent emerging challenges to ensure functional and quality documentation [26]. Figure 4 summarizes the challenges to software documentation in FOSS extracted and categorized from our primary studies, namely **Collaboration**, **Quality**, **Incompleteness**, **Maintainability**, and **Categorization**. These challenges are discussed below.

Collaboration in software documentation for FOSS projects encounters challenges from distributed teams, hindering real-time communication and causing delays [1, 16]. Inconsistencies arise from varying expertise levels, conflicting priorities, and dynamic development necessitates continuous revision. Sun et al. [60] {PS10} stated that a better understanding of developer collaboration can lead to the development of superior artifacts. However, beyond identifying collaborative patterns in FOSS projects, the authors emphasized that the most collaborative type of files are test files, with documentation taking a secondary role in collaborative work. The rationale is that test files involve multiple stakeholders and provide immediate feedback. In turn, Puhlfürß et al. [54] {PS09} explained that some popular projects use GitHub to make their code public rather than to develop a collaborative community that prioritizes comprehensive publicly available documentation.

The **Quality** of software documentation is a multifaceted challenge. However, to effectively impact software development, quality assurance must systematically address software documentation quality as well [52]. Aversano et al. [8] {PS04} conducted a case study to assess the quality of documentation in open-source systems to understand the support it can offer. According to the authors, the results indicate that documentation is not always available and only partially meets developers' needs, often being incorrect, incomplete, outdated, and ambiguous. Conversely, Carvalho et al. [11] {PS01}

provide a discussion into the difficulty of assessing documentation quality due to the subjectivity involved. Their work described three characteristics that directly impact overall documentation quality: i) Readability: The text's readability can be subjective, but there are linguistic characteristics that generally hinder reading. Some can even be measured, such as the number of syntax errors or excessive use of abbreviations; ii) Timeliness: This is an important documentation characteristic and for other textual files; they must be up-to-date and refer to the software's latest version; and iii) Completeness: this characteristic indicates how comprehensive the documentation is and if it covers all necessary topics.

The documentation **Incompleteness** arise if the documentation does not contain the information about the system needed by practitioners/users to perform their tasks [69]. Ding et al. [20] {PS02} argue that instead of collecting and documenting FOSS project information, developers prefer to use forums, discussion lists, and social media to obtain information and thus expend less effort. According to the authors, architecture documentation is not widely created and used in FOSS projects. Furthermore, documentation related to refactoring is also uncommon in open-source software. However, according to AlOmar et al. [4] {PS07}, providing sufficient documentation related to software refactorings is essential to facilitate code review processes. Additionally, Prana et al. [53] {PS06} introduces that a repository's README files on GitHub are usually the first documents developers access, but there is no systematic standard regarding the types of documents exposed in README files, contributing to the lack of documents or relevant content.

Moreover, documentation **Maintainability** concerns issues related to how easy it is to apply changes or corrections to it [2]. Bigliardi et al. [9] {PS11} stated that each open-source project has its peculiarities, but some show a positive correlation with maintenance effort as the system evolves, while others show the opposite behaviour. From a quantitative perspective, the authors observed that a significant portion of artefacts not related to source code does not evolve alongside the FOSS project.

Lastly, **Categorizing** different types of documentation has proven to be challenging, as described in the existing literature. More specifically, this challenge refers to the difficulty in organizing and classifying various documents within a software development context, such as user manuals, technical specifications, API documentation, etc [3]. Ma et al. [40] {PS05} investigated how artifacts can be categorized and what types of artifacts are created during FOSS development. The authors emphasized that categorizing documentation can provide insights into software projects from both technical and managerial perspectives. Additionally, Wermke et al. [65] {PS11} highlighted that perspectives on documentation provision seem to depend on the team's context; support for documentation appears to be correlated with team size and the number of less experienced developers. Following a specific formatting and well-defined structure style, code comments also serve as a type of documentation that can provide significant support to developers [14].

4.3 Solutions

According to our analysis, strategic pathways have been identified as solutions in the context of FOSS projects. This perspective aligns with the idea that open-source communities need a concise view of different types of actions to select viable and suitable for their needs and the challenges they face [62]. The three solutions (**Strategic Use of README**, **Adoption of Artificial Intelligence**, and **Support Tools & Approaches**) explored by primary studies are summarized in Figure 5, which will be discussed below.

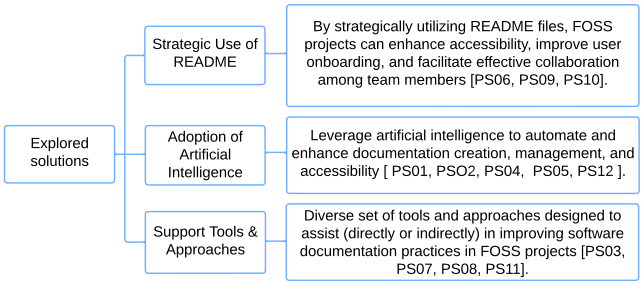

Figure 5: Summary of the explored solutions.

The **Strategic Use of README** file in FOSS projects has emerged as a potential solution that is worth highlighting. These files are often included to provide essential information about the project, such as instructions for installation, usage guidelines, and key features [37]. By utilizing README files strategically, software projects can enhance accessibility, improve user onboarding, and facilitate effective collaboration among team members [39].

Sun et al. [60] {PS10} investigated the collaboration frequency in different types of files in open-source software projects using Author Cross Entropy (ACE) and leveraging data from the World of Code dataset and the GitHub API. Since file types vary in size and characteristics, the Mann-Whitney U classification was used to compare the ACE distribution across each file type. Upon analyzing the commits and README files, they found that test files and documentation manifest a notable degree of collaboration. According to the authors, collaboration on documentation in the GitHub platform is a positive bias, as a versatile degree of change commits can be observed in the documentation. Furthermore, authors often add comments to explain their changes after modifying documentation to reinforce collaboration with team members.

Puhlfürß et al. [54] {PS09}, through exploratory content analysis, found that a structured README on GitHub facilitates collaboration. Contributors prefer placing all documentation in README files if they need to document many pages. GitHub features such as issue tracker, wiki, and pull requests are used to create subsequent connections in the project. Another factor is that the studied projects used combinations of various types of textual artifacts to document product features, demonstrating each contributor's preferences. However, the authors conclude that the lack of linkage between documentation and source code can limit understanding and project maintenance capability over time. Thus, a straightforward approach to achieving better linkage between documentation, code, and promoting collaboration would be a section at the top of the README file describing the project structure, and a best practice would be using hyperlinks to facilitate efficient navigation between product feature documentation and source code.

Additionally, Prana et al. [53] {PS06} proposed an approach for automated classification of GitHub README content. The built

classifier explored a binary relevance method for multi-label classification, where each multi-label classification problem is transformed into a set of binary classifications. Authors used linguistic patterns, single-word header in non-English languages, repository name, and non-ASCII content. The most prevalent categories in the classification were: i) 'What' category based on titles like "Introduction" and "About" providing a brief introduction to the project; ii) 'How' category, including instructions on how to use the project, with programming-related content (*e.g.*, setup, installation, dependencies, and errors/bugs), and iii) 'Who' category including information about licenses, contact details, and code of conduct. Based on the sample provided by the authors, 97% of files contain at least one section describing the 'What' of the repository, and 88.5% provide some 'How' content. In addition to automatically classifying content, the classifier can allow access to unstructured information contained in a GitHub README and visually label artifact types based on the information contained in them.

On the other hand, the **Adoption of Artificial Intelligence** has been also emerged as a possible trend in software documentation for FOSS projects. This trend involves leveraging artificial intelligence to automate and improve various aspects of documentation creation, management, and accessibility. Carvalho et al. [11] {PS01} introduced a data mining-based approach that provides a systematic tool for collecting metrics about software documentation content and evaluating its quality. Quality metrics are processed according to pre-established characteristics by the authors, such as readability, timeliness, and completeness of documentation. Through FOSS Documentation Mining (DMOSS), it was possible to extract insights and useful information regarding documentation. The authors began the study by collecting content written in natural language, processing it to calculate metrics, and finally analyzing these metrics to draw conclusions. Data is analyzed from the Annotated Tree structure. In this tree, nodes represent files and directories, and edges describe the hierarchical structure of the package. The tree is generated automatically by DMOSS, functioning by recursively traversing the file system hierarchy, adding nodes for each file and directory. For each node that has a documentation format, simple text content is extracted and added to the corresponding node as an attribute. DMOSS's main contribution is to provide structured documentary analysis that can be used to implement and integrate new metrics and analyze documentation rapidly. Thus, it sought to provide information about the content found in artifacts and assess quality according to an ordered understanding of the knowledge granted in artifacts.

Aversano et al. [8] {PS04} delved into the evaluation of software document quality across various types using metrics and quality indicators. They highlighted that assessing these metrics demands the application of Natural Language Processing, Information Extraction, and Information Retrieval techniques. The evaluated metrics were i) Completeness: this indicator verifies if the documentation describes all items (packages, classes, methods) in the source code; ii) Alignment: checks if the documentation is up to date with the current release; iii) Dimension: analyzes if document phrases are too long or too short, aiming to verify if the text is too difficult to understand; iv) Structure: aims to evaluate the structure in terms of number of chapters, sections, sub-section nesting, document length, and density of tables and figures, and v) Readability: examines if sentences express clear and understandable concepts. On the other hand, Ciurumelea et al. [14] {PS12} used neural language models to analyze structural elements in documentation comments to evaluate how code comments can assist developers in writing documentation. The models used were the Sequence LM Model, Context LM Model, Section Context LM Model, and Section-Specific Context LMs. The models were evaluated using the Top-k technique to test their performance. According to the authors' analyses, documentation comments are widely used, especially in Python and Java projects. Thus, having structural elements improves artefact readability and allows automated processing to generate documentation or provide metrics that support development environments.

Ma et al. [40] {PS05} proposed an automated approach based on Machine Learning techniques to identify various types of software artifacts. Decision tree approaches, support vector machines, and Bayesian networks were used. Using heuristics, the authors categorized artifacts into two groups: those that can be classified based solely on file name and extension (*e.g.*, bat files, and those that require a deeper analysis to be classified, such as text documents. Thus, the results of the empirical study indicate that the automated approach based on Machine Learning techniques was able to automatically classify software artifacts with an average accuracy of 85% and recall of 82% using 10-fold cross-validation on the validation dataset. Additionally, it was shown that besides source code, about 14.88% of open-source projects contain other forms of artifacts, such as requirements documents and architecture documents, which are of interest to software engineering researchers.

Regarding architecture documentation, Ding et al. [20] {PS02} clarified that this type of documentation is essential to promote anarchic collaboration while preserving centralized control over interfaces in successful open-source projects. The authors concluded, from an exploratory survey, that the likelihood of an FOSS project documenting software architecture increases with the number of developers involved, suggesting that architectural documentation seems more useful for larger projects. Thus, the amount of architecture documentation largely depends on contextual development factors, with architectural element models, system, and project mission as frequently documented architectural information.

Finally, a category of solution focused on **Support Tools & Approaches** has been identified. This category encompasses tools and approaches designed to assist in improving software documentation practices. AlOmar et al. [4] {PS07} conducted an exploratory study on how developers document their refactoring activities in commit messages. They used the Refactoring Miner tool to identify refactoring operations occurring in projects. The Refactoring Miner iterates over a repository's commit history chronologically and compares changes made to source code files to detect refactorings. They found that developers tend to use a variety of textual patterns to document their refactoring activities, such as refactor, move, and extract. Automatically classifying a large set of commits containing refactoring activities triggers motivations beyond the basic need for system documentation improvement, as it demonstrates guiding practices for refactoring activities that include the following categories: Functional, Bug Fixing, Internal Quality Attribute, Code Smell Resolution, and External Quality Attribute. The authors

explain that contributors who frequently refactor code tend to document changes less than developers who refactor occasionally.

Pasuksmit et al. [46] {PS08} proposed a prediction model called DocWarn to estimate the probability of changes in documentation during the sprint time. Applying DocWarn's semantic difference criteria, based on authors' qualitative evaluation, 40% to 68% of predicted documentation changes are related to scope modification. With DocWarn's probability estimation, the team will be more aware of possible documentation changes during sprint planning, allowing the team to manage uncertainty and reduce the risk of erroneous effort and planning estimates. Wermke et al. [65] {PS11} complements that the quality and available documentation for FOSS software are often directly correlated with project popularity. Furthermore, more popular projects have higher rates of hours dedicated to documentation maintenance and creation.

Bigliardi et al. [9] {PS03} conducted a quantitative study to investigate whether the maintenance effort in documentation increases as the system evolves. They analyzed the ranking correlation between project age (i.e., time) and evolution of commit percentage on artifacts using Shapiro-Wilk's normality test and the Gini coefficient for data analysis. The analysis was done on all commits from oldest to newest, whether in updates or removal of files. They found that the behaviour is inherently relative to the type of project, with maintenance effort increasing over time in some and not in others.

5 DISCUSSION

In this SMS, we mapped the challenges and solutions concerning the documentation in FOSS over papers published in the past decade. As discussed by Aghajani et al. [3] and Ding et al. [18], software documentation provides developers and users with a description of what a system does, how it operates, and how it should be used. Thus, it is critical to comprehend the challenges and solutions associated with documentation in FOSS projects within their context.

The identified challenges encompassed a range of factors leading to difficulties faced by FOSS projects. We detail in Figure 4 the documentation challenges depicted by our primary studies, which include **Collaboration**, **Quality**, **Incompleteness**, **Maintainability**, and **Categorization**. Each challenge has nuances, but they all converge when discussing the role of documentation in the FOSS processes. It is noteworthy that documentation also contributes to recording software evolution, laying the groundwork for subsequent processes such as training, usage, and maintenance [22, 44].

The **Collaboration** reflected well-known particularities faced by FOSS, including dispersed teams, diverse skill levels, and the need to attract and retain new contributors [1, 16]. This intricate dynamism of software development may led to continuous revisions due to conflicting priorities and varying expertise levels, resulting in document inconsistencies [60]. While collaborative patterns primarily favoured test files over documentation in FOSS, the importance of documentation in fostering collaborative communities remained understated [54]. **Quality** issues in documentation also constituted another challenge. Despite being an indispensable resource, documentation often fell short of meeting developers' needs due to inaccuracies and ambiguity [8]. In this sense, assessing documentation quality proved intricate owing to its subjective nature and involve multiple factors (readability, timeliness, etc) [11, 52].

The challenge of **Incompleteness** accentuated the deficiency of essential information in the documentation required for practitioners' tasks [69]. Indeed, forums and social media became alternative sources of information for developers, but the scarcity of comprehensive architecture and refactoring documentation persisted as an open issue in FOSS projects [4, 20]. Additionally, the lack of systematic standards for README files evidences the scarcity of relevant documentation [53]. The **Maintenance** perspective also highlighted the need to provide ease of making changes or corrections in documents [2]. While some projects exhibited positive correlations between maintenance effort and documentation evolution, others did not [9]. The **Categorization** of documentation types also proved to be relevant by laying in organizing and classifying diverse documents like user manuals, technical specifications, and API documentation, reflecting the multifaceted nature of documentation in FOSS projects [3, 40]. Furthermore, the level of documentation support appeared to correlate with team size and experience levels, underscoring the contextual dependencies [65].

The solutions emerged by our primary studies (see Figure 5) include the **Strategic Use of README**, **Adoption of Artificial Intelligence**, and **Support Tools & Approaches**. These solutions demonstrate the ongoing efforts to enhance documentation practices and support efficient collaboration and maintenance in FOSS initiatives. In particular, the **Strategic Use of README** files in FOSS projects emerged as a artifact to pay attention. These files typically contain information about the project, such as installation instructions, usage guidelines, and key features [37]. By strategically utilizing README files, FOSS projects can enhance accessibility, improve user onboarding, and facilitate effective collaboration among team members [39].

Another perspective of solution gaining traction stems from the **Adoption of Artificial Intelligence**. This trend involves leveraging artificial intelligence to automate and enhance documentation creation, management, and accessibility. Ciurumelea et al. [14] {PS12}, for example, explored neural language models to analyze structural elements in documentation comments to evaluate how code comments can assist developers in writing documentation. Lastly, **Support Tools & Approaches** have been identified as instrumental in improving software documentation practices in FOSS projects. AlOmar et al. [4] {PS07} conducted an exploratory study on developers' documentation of refactoring activities in commit messages with the support of the Refactoring Miner tool. Additionally, Pasuksmit et al. [46] {PS08} proposed DocWarn, a prediction model estimating the probability of documentation changes during sprint time, aiding in effort estimation and risk reduction.

When considering the relationship between the challenges and solutions, we may observe that Strategic use of README can help streamline the collaboration process, for example, by facilitating the onboarding of newcomers or by serving as a summary of document categories. Additionally, the Adoption of Artificial Intelligence could help generate documentation automatically by analyzing the code, and subsequently improve the quality of documentation and help to keep it up to date, overcoming incompleteness. Finally, the Support Tools & Approaches may enhance documentation maintenance through automation, version control integration, collaborative editing, template standardization, promoting best practices, etc.

For instance, automated tools like Sphinx or Javadoc can update documentation based on code changes, reducing manual effort.

As we can see, the findings from this work have implications beyond academic context and may be helpful in the software industry. Understanding the challenges and solutions can help FOSS stakeholders make informed decisions on prioritizing documentation efforts, encouraging collaboration and utilizing efficient approaches. Hopefully, these outcomes may contribute to promoting concrete reflection and bringing awareness to the documentation process in FOSS projects and, consequently, lead efforts to improve the quality, accessibility, and sustainability of software documentation practices in line with FOSS evolving demands and complexities.

6 THREATS TO VALIDITY

To properly clarify our validity threats and corresponding mitigation actions, we explored the checklist proposed by Ampatzoglou et al. [6] to identify threats in secondary studies in SE research.

Study inclusion/exclusion bias: This work recognizes that there may be other papers on software documentation in FOSS that were not included in the sampling frame explored. These papers may fall outside the publication period between 2013 and 2023 or may not use terms included in the search string. However, measures were taken to mitigate this threat. Initially, we explicitly justified this timeframe as strategic since it allows us to focus on recent perspectives. Moreover, this decision is clearly stated in the research question. The period between 2013 and 2023 was specified in the filters of each database used in the search, considering the focus of this SMS. Even with this restriction, the considerable number of papers (1271) obtained after executing the search is worth noting. We also crafted a search string combining various terms and synonyms related to FOSS and software documentation. This issue included different variations like `open source` and `open-source`, as well as acronyms like `OSS` and `FOSS`. Terms related to documentation or documents were also included. Furthermore, we used backward and forward snowballing processes to identify potential missed studies.

Researcher Bias and Repeatability: The search process rigorously explored our search string across different well-known databases, thereby enhancing research repeatability and transparency. The first author conducted the initial search in each database and removed duplicate papers. In the application of Inclusion Criteria (IC) and Exclusion Criteria (EC), the full paper list was divided into two groups, and other two co-authors applied IC and EC, while the main author conducted a double check on all papers (from both groups). Researchers met virtually to resolve decision conflicts and reached a preliminary selection consensus. Two more experienced co-authors were invited to discuss the final decision if conflicts persisted. This process contributed to enhancing data accuracy.

Robustness of Classification: Our qualitative analysis of challenges and solutions may face a potential threat due to inherent subjectivity. Although the analysis was based on findings from selected papers, the categorizing process is influenced by the authors' interpretations, which is natural and valuable in qualitative research [21]. However, this open coding process underwent discussions among the first author and two other co-authors with experience SE research to strengthen understanding and rigor.

7 CONCLUSION

This paper investigated the challenges and solutions associated with software documentation in Free and Open Source Software (FOSS). Through a Systematic Mapping Study (SMS) focused on the papers published between 2013 and 2023, a search string was explored to select primary studies, supplemented by backward snowballing. Our search strategy yielded a total of 12 primary studies. The findings unveiled five major challenges (Collaboration, Quality, Incompleteness, Maintainability, and Categorization) and three overarching solution perspectives (Strategic Use of README, Adoption of Artificial Intelligence, and Support Tools & Approaches) pertinent to documentation in FOSS. These findings helped us to answer our research question.

This study makes novel contributions to both academia and practice. From an academic perspective, this study offers a SMS that reveals a set of challenges and solutions related to software documentation, a topic that has been relatively underexplored in FOSS despite its significant relevance. As far as we know, no other paper has addressed this objective yet. Our study also offers a comprehensive perspective of the research topic under investigation, presenting existing knowledge and highlighting research gaps. Moreover, our results evidence valuable opportunities for future research. From a practical standpoint, this work promotes a reflective examination of the use of documentation in FOSS, highlighting challenges and solutions that can contribute to enhancing documentation quality process. This discussion can aid in understanding the challenges and solutions that promote increased documentation quality and project efficiency in FOSS projects.

Future work could investigate the applicability of the challenges and solutions identified in our SMS to GitHub projects. This investigation could provide valuable findings into how these challenges and solutions manifest and are addressed within a platform that hosts diverse software projects. In addition, future research could eventually explore the contrasts and similarities in software documentation practices between closed-source and FOSS projects.

ARTIFACTS AVAILABILITY

All data supporting this study are openly available through our supporting repository [50], ensuring transparency, replicability, and accessibility in regard to our Systematic Mapping Study protocol.

REFERENCES

[1] Mark Aberdour. 2007. Achieving quality in open-source software. *IEEE software* 24, 1 (2007), 58–64.
[2] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C Shepherd. 2020. Software documentation: the practitioners' perspective. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 590–601.
[3] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1199–1210.
[4] Eman Abdullah AlOmar, Anthony Peruma, Mohamed Wiem Mkaouer, Christian D Newman, and Ali Ouni. 2021. Behind the Scenes: On the Relationship Between Developer Experience and Refactoring. *arXiv e-prints* (2021), arXiv–2109.
[5] Scott W Ambler. 2001. Agile Documentation. 2001-2004. *The Official Agile Modeling (AM) Site* (2001).
[6] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. 2019. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software*

Technology 106 (2019), 201–230.

[7] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. 2019. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[8] Lerina Aversano, Daniela Guardabascio, and Maria Tortorella. 2017. Evaluating the quality of the documentation of open source software. In *International Conference on Evaluation of Novel Approaches to Software Engineering*, Vol. 2. SciTePress, 308–313.

[9] Luca Bigliardi, Michele Lanza, Alberto Bacchelli, Marco D'Ambros, and Andrea Mocci. 2014. Quantitatively exploring non-code software artifacts. In *2014 14th International Conference on Quality Software*. IEEE, 286–295.

[10] Lionel C Briand. 2003. Software documentation: how much is enough?. In *Seventh European Conference onSoftware Maintenance and Reengineering, 2003. Proceedings.* IEEE, 13–15.

[11] Nuno Ramos Carvalho, Alberto Simoes, and José Joao Almeida. 2014. DMOSS: Open source software documentation assessment. *Computer Science and Information Systems* 11, 4 (2014), 1197–1207.

[12] Chapin. 2000. Trends in preserving and enhancing the value of software. In *Proceedings 2000 International Conference on Software Maintenance*. IEEE, 6–8.

[13] Frank A Cioch, Michael Palazzolo, and Scott Lohrer. 1996. A documentation suite for maintenance programmers. In *1996 Proceedings of International Conference on Software Maintenance*. IEEE Computer Society, 286–286.

[14] Adelina Ciurumelea, Carol V Alexandru, Harald C Gall, and Sebastian Proksch. 2023. Completing Function Documentation Comments Using Structural Information. *Empirical Software Engineering* 28, 4 (2023), 86.

[15] Hilda Simone Coelho. 2009. Documentação de software: uma necessidade. *Texto Livre: linguagem e tecnologia* 2, 1 (2009), 17–21.

[16] Kevin Crowston, Qing Li, Kangning Wei, U Yeliz Eseryel, and James Howison. 2007. Self-organization of teams for free/libre open source software development. *Information and software technology* 49, 6 (2007), 564–575.

[17] 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*. 68–75.

[18] Wei Ding, Peng Liang, Antony Tang, and Hans van Vliet. 2014. Knowledge-based approaches in software documentation: A systematic literature review. *Information and Software Technology* 56, 6 (2014), 545–567. https://doi.org/10.1016/j.infsof.2014.01.008

[19] Wei Ding, Peng Liang, Antony Tang, and Hans Van Vliet. 2014. Knowledge-based approaches in software documentation: A systematic literature review. *Information and Software Technology* 56, 6 (2014), 545–567.

[20] Wei Ding, Peng Liang, Antony Tang, Hans Van Vliet, and Mojtaba Shahin. 2014. How do open source communities document software architecture: An exploratory survey. In *2014 19th International conference on engineering of complex computer systems*. IEEE, 136–145.

[21] Tore Dybå, Rafael Prikladnicki, Kari Rönkkö, Carolyn Seaman, and Jonathan Sillito. 2011. Qualitative research in software engineering. *Empirical Software Engineering* 16 (2011), 425–429.

[22] Andrew Forward. 2002. *Software documentation: Building and maintaining artefacts of communication.* University of Ottawa (Canada).

[23] Oscar Franco-Bedoya, David Ameller, Dolors Costal, and Xavier Franch. 2017. Open source software ecosystems: A Systematic mapping. *Information and software technology* 91 (2017), 160–185.

[24] Felipe Fronchetti, David C. Shepherd, Igor Wiese, Christoph Treude, Marco Aurélio Gerosa, and Igor Steinmacher. 2023. Do CONTRIBUTING Files Provide Information about OSS Newcomers' Onboarding Barriers?. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (, San Francisco, CA, USA,) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 16–28. https://doi.org/10.1145/3611643.3616288

[25] Golara Garousi. 2012. *A Hybrid Methodology for Analyzing Software Documentation Quality and Usage.* Master's thesis. Graduate Studies.

[26] 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*. 24–35.

[27] Golara Garousi, Vahid Garousi-Yusifoğlu, 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 (2015), 664–682.

[28] Basit Habib and Rohaida Romli. 2021. A systematic mapping study on issues and importance of documentation in agile. In *2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 198–202.

[29] James D. Herbsleb and Audris Mockus. 2003. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on software engineering* 29, 6 (2003), 481–494.

[30] Leila Lage Humes. 2007. Communities of practice for open source software. In *Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives*. IGI Global, 610–623.

[31] Zilia Iskoujina and Joanne Roberts. 2015. Knowledge sharing in open source software communities: motivations and management. *Journal of Knowledge Management* (2015).

[32] Md Athikul Islam, Rizbanul Hasan, and Nasir U Eisty. 2023. Documentation Practices in Agile Software Development: A Systematic Literature Review. In *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*. IEEE, 266–273.

[33] Pankaj Jalote. 2012. *An integrated approach to software engineering*. Springer Science & Business Media.

[34] Rajdeep Kaur, Kuljit Kaur Chahal, and Munish Saini. 2022. Understanding community participation and engagement in open source software Projects: A systematic mapping study. *journal of king saud university-computer and information sciences* 34, 7 (2022), 4607–4625.

[35] Barbara Kitchenham, Stuart Charters, et al. 2007. Guidelines for performing systematic literature reviews in software engineering version 2.3. *Engineering* 45, 4ve (2007), 1051.

[36] Daniel Klug, Christopher Bogart, and James D Herbsleb. 2021. " They Can Only Ever Guide" How an Open Source Software Community Uses Roadmaps to Coordinate Effort. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–28.

[37] Miika Koskela, Inka Simola, and Kostas Stefanidis. 2018. Open source software recommendations using github. In *Digital Libraries for Open Knowledge: 22nd International Conference on Theory and Practice of Digital Libraries, TPDL 2018, Porto, Portugal, September 10–13, 2018, Proceedings 22*. Springer, 279–285.

[38] Modi Lakulu, Rusli Abdullah, Mohd Hasan Selamat, Hamidah Ibrahim, and Mohd Zali Mohd Nor. 2010. A framework of collaborative knowledge management system in open source software development environment. *Computer and Information Science* 3, 1 (2010), 81.

[39] Yuyang Liu, Ehsan Noei, and Kelly Lyons. 2022. How ReadMe files are structured in open source Java projects. *Information and Software Technology* 148 (2022), 106924.

[40] Yuzhan Ma, Sarah Fakhoury, Michael Christensen, Venera Arnaoudova, Waleed Zogaan, and Mehdi Mirakhorli. 2018. Automatic classification of software artifacts in open-source applications. In *Proceedings of the 15th International Conference on Mining Software Repositories*. 414–425.

[41] PA Michelazzo. 2008. Documentação de software, 2006.

[42] Martin Michlmayr, Francis Hunt, and David Probert. 2005. Quality practices and problems in free software projects. In *Proceedings of the first international conference on open source systems*. 24–28.

[43] Martin Michlmayr, Francis Hunt, and David Probert. 2007. Release management in free software projects: Practices and problems. In *Open Source Development, Adoption and Innovation: IFIP Working Group 2.13 on Open Source Software, June 11–14, 2007, Limerick, Ireland 3*. Springer, 295–300.

[44] Vanessa B Nunes, Andrea O Soares, and Ricardo A Falbo. 2004. Apoio à Documentação em um Ambiente de Desenvolvimento de Software. In *Memorias de VII Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software-IDEAS*. 50–55.

[45] David Lorge Parnas. 2010. Precise documentation: The key to better software. In *The future of software engineering*. Springer, 125–148.

[46] Jirat Pasuksmit, Patanamon Thongtanunam, and Shanika Karunasekera. 2022. Towards reliable agile iterative planning via predicting documentation changes of work items. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 35–47.

[47] Bruce Perens et al. 1999. The open source definition. *Open sources: voices from the open source revolution* 1 (1999), 171–188.

[48] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*. 1–10.

[49] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and software technology* 64 (2015), 1–18.

[50] Giniele Pinho, Aguiar Jeová Caçula, Lucas Costa, Igor Wiese, and Allysson Allex Araújo. 2024. Challenges and Solutions of Free and Open Source Software Documentation: A Systematic Mapping Study - Supporting repository. https://doi.org/10.5281/zenodo.12953652

[51] Victor Hugo Miranda Pinto. 2021. O papel da documentação no desenvolvimento de software open source: Uma análise e um estudo de caso. (2021).

[52] Reinhold Plösch, Andreas Dautovic, and Matthias Saft. 2014. The value of software documentation quality. In *2014 14th International Conference on Quality Software*. IEEE, 333–342.

[53] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the content of github readme files. *Empirical Software Engineering* 24 (2019), 1296–1327.

[54] Tim Puhlfürß, Lloyd Montgomery, and Walid Maalej. 2022. An Exploratory Study of Documentation Strategies for Product Features in Popular GitHub Projects.

In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 379–383.
[55] Johnny Saldaña. 2021. The coding manual for qualitative researchers. (2021).
[56] David M Schweiger, William R Sandberg, and James W Ragan. 1986. Group approaches for improving strategic decision making: A comparative analysis of dialectical inquiry, devil's advocacy, and consensus. *Academy of management Journal* 29, 1 (1986), 51–71.
[57] André Santiago da Fonseca Silva. 2020. *Documentação de software: uma análise comparativa entre documentação tradicional e living documentation*. Master's thesis. Universidade Federal do Rio Grande do Norte.
[58] Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L Bleris. 2002. Code quality analysis in open source software development. *Information systems journal* 12, 1 (2002), 43–60.
[59] Igor Steinmacher, Ana Paula Chaves, Tayana Uchoa Conte, and Marco Aurélio Gerosa. 2014. Preliminary empirical identification of barriers faced by newcomers to Open Source Software projects. In *2014 Brazilian Symposium on Software Engineering*. IEEE, 51–60.
[60] Weijie Sun, Samuel Iwuchukwu, Abdul Ali Bangash, and Abram Hindle. 2023. An Empirical Study to Investigate Collaboration Among Developers in Open Source Software (OSS). In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 352–356.
[61] Theo Theunissen, Uwe van Heesch, and Paris Avgeriou. 2022. A mapping study on documentation in Continuous Software Development. *Information and software technology* 142 (2022), 106733.
[62] Bianca Trinkenreich. 2021. Please Don't Go—A Comprehensive Approach to Increase Women's Participation in Open Source Software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 293–298.
[63] Joey Van Angeren, Jaap Kabbedijk, Slinger Jansen, and Karl Michael Popp. 2011. A Survey of Associate Models used within Large Software Ecosystems.. In *IWSECO@ ICSOB*. 27–39.
[64] Georg Von Krogh and Eric Von Hippel. 2006. The promise of research on open source software. *Management science* 52, 7 (2006), 975–983.
[65] Dominik Wermke, Jan H Klemmer, Noah Wöhler, Juliane Schmüser, Harshini Sri Ramulu, Yasemin Acar, and Sascha Fahl. 2023. " Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1545–1560.
[66] Joel West and Marcel Bogers. 2014. Leveraging external sources of innovation: A review of research on open innovation. *Journal of product innovation management* 31, 4 (2014), 814–831.
[67] Titus Winters, Tom Manshreck, and Hyrum Wright. 2020. *Software engineering at google: Lessons learned from programming over time*. O'Reilly Media.
[68] Ming-Wei Wu and Ying-Dar Lin. 2001. Open Source software development: An overview. *Computer* 34, 6 (2001), 33–38.
[69] Junji Zhi, Vahid Garousi-Yusifoğlu, 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.