# *Exploring the Impact of Open-Source Software Documentation on Newcomers' Motivation and Participation*

AALBORG
UNIVERSITET

Peter Szatmary
Maciej Karol Schweitzer
CS-IT
Aalborg Universitet
Januar 5th 2023

**Computer Science**

Selma Lagerløfs Vej 300
9220 Aalborg
http://www.aau.dk

**Title:**

Exploring the Impact of Open-Source Software Documentation on Newcomers' Motivation and Participation

**Project period:**

2022.09.01 - 2023.01.05

**Authors:**

Peter Szatmary
Maciej Karol Schweitzer

**Supervisor:**

Adam Alami

**Edition: 1**
**Number of pages: 31**
**Completed: 2023.01.05.**

# Preface


## Abstract

This report examines the role of open-source software in facilitating knowledge transfer within communities of developers and the barriers that newcomers to open-source projects may face, with a focus on documentation issues and how they relate to the motivation of newcomers.

After setting the scene with the introduction of open-source and related concepts, first a review of relevant literature is presented, highlighting the importance of open-source software and the challenges that newcomers can encounter when participating in open-source projects.

The research method for the study, which will be conducted next semester is described as a case study with qualitative data analysis, using thematic analysis to identify key themes and patterns in the data. The findings of this study will answer our research question deepening our understanding of experiences and motivation of newcomers in open-source projects and the ways in which they can be supported in overcoming challenges they face, thereby giving more insight into the dynamics of knowledge transfer in open-source communities.

# Table of Contents

# Introduction 1

## 1.1 Open-source software

One way of classifying computer programs is by the availability of its source code that later on gets interpreted or compiled to binary files that actually run on our various devices. This way we a program source code can be either closed or open-source.

Closed source software, typically but not exclusively is made by firms and corporations eventually selling them to customers either standalone or as part of a larger stack of software. Closed source software as its name implies reserve some or all rights regarding end users being able to look, modify or resell works based on it legally.

In the case of the latter, there are numerous definitions of it circulating around; the common umbrella term used is Free and open-source software (FOSS) which incorporates two similar but far from equivalent concepts as well as definitions, namely free software and open-source software [GNU, 2022]. At their core, both promote similar values such as under their terms granting users - anyone rights to freely use, study, modify, improve or even profit off of them, commercialising and selling it, its modified versions or programs incorporating it under their name or company. Various licenses clearly define the scopes of these, but we opt not to expand upon them, as it is not the main focus of this thesis [OSI, 2022a]. The main difference, according to Richard M. Stallman [Stallman, 1998], the founder of the GNU project and the Free Software Foundation (FSF) is that open-source is a development methodology, while free software is a social movement. The GNU project further explains it by saying *"Thus, 'free software' is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech', not as in 'free beer.'"* [Stallman, 1998].

In the subsequent parts of our research and report, we will focus on open-source software as a development methodology based on open collaboration. Anyone, from any country of the world can contribute to various open-source projects asynchronously, not without being bound to standard workday hours. However, this does not mean that it is easy or that all contributions will be blindly accepted.

Open-source software is important because it promotes the open development of powerful software tools on which we are increasingly depending. Some great examples of well-known open-source software are the Linux family of operating systems (including the Android mobile

operating system), the Mozilla Firefox or the Chromium web browser. It presents interesting challenges to the developer community, but also allows for the development of problem-solving skills and the opportunity to contribute back to the community. [OSI, 2022b].

Access to open source projects are generally available through many different repositories, often on self- or internet-hosted Git services like GitHub or GitLab. These repositories usually contain contribution guidelines, issue trackers, and documentation if they exist. A contributor, but even the project author or owner one can expect feedback or suggestions from others on development of new features, bug fixing, or writing documentation or know-how guides.

## 1.2 Scope of the topic

Learning about new technologies on one's own can be difficult. Working on an open-source project can provide the opportunity to expand one's knowledge working on it, while also giving back to the community. This work makes it possible to learn from other, possibly more experienced developers in a particular area. It is an important process for both parties, as both gain and share knowledge. In the context of open-source communities, the authors of a project give other people access to view and edit its parts. As a result, all documentation related to the project becomes important in the process of knowledge transfer.

Unfortunately, just as any software project, open source projects include many obstacles [Balali et al., 2018]. Starting with abandonment, lack of documentation, missing issue trackers and contribution guidelines and so much more. Another big problem is the quality of knowledge transfer and the tools used for that end [Steinmacher et al., 2015].

As is the case with closed-source enterprise software made for profit, an open-source software project have to survive somehow as well. It should have enough members to support it, fix bugs, develop new features.

However, while usually companies are paying for the developers' time to write their software making it easier to attract new developers to work for them, this is not typically the case with open-source projects. Consequently, many have problems both with attracting newcomers and with the retention of contributors. This process becomes even more challenging as a project becomes bigger and older. Indeed as senior developers leave a project they may not transfer their domain knowledge. Also, due to globally distributed nature of open-source software, all communication has to be done online compared to the usual for-profit enterprises situatedness in-office entailing face-to-face communication that is associated with better knowledge transfer. Working in a dispersed team or community usually works asynchronously, it is extremely important to maintain common, clear communication and the idea in which the project will develop, and that in the case of OSS is some form of shared documentation [Ågren et al., 2022]. Knowledge being lost by not being written down, documented somewhere can and as we will

see is a problem in open-source projects.

Lacking docs is also a form of technical debt [Li et al., 2015]. Technical debt in software refers to the cost associated with making quick and dirty decisions during the development process that make it easier to get the software working in the short term, but which may create problems or additional work in the long term. This can include taking shortcuts in the code, using suboptimal algorithms, or skipping important tasks like writing documentation. These decisions can save time and effort in the short term, but they can create problems later on when the code needs to be maintained or updated. This can make the code more difficult to understand and work with, which can increase the time and effort required to make changes or add new features [Li et al., 2015].

# Motivation 2

## 2.1 Motivation

Our motivation also originates from our personal experiences trying to use and contribute to open-source projects. In former semesters, we've been working extensively with various open-source frameworks to solve a given task. After the deadlines, having finished the projects, we always made time to evaluate and discuss on what should have been better, what could have been improved and what could be avoided in the future. One prominent common point that was repeatedly brought up was various feelings about understanding the structure and the codebase of the open-source projects.

Some included detailed, well-rounded instructions, while others included a lot less. For one specific semester, we worked with an automated planning system using various machine learning techniques and their associated repositories. Unfortunately, that project took significantly more time than anticipated. One of the major differences compared to some other projects before was the very limited amount of documentation in addition to unclear code and lack of in-code comments. The lack of proper documentation made it extremely time-consuming to comprehend the codebase. These experiences ultimately led us to the conclusion that a further examination of knowledge transfer and documentation in open-source projects would be valuable.

## 2.2 Research question

The prospect of working with open source code has many advantages and can tempting to do: for professional development, personal portfolio improvement, networking opportunities or simply just giving back to the community [Nagle, 2016]. However, potential contributors frequently encounter barriers or obstacles trying to do so [Balali et al., 2018] preventing them from effectively participating in these projects. One such barrier is inadequate documentation [Aghajani et al., 2019].

This study aims to investigate the specific role that documentation plays in this process, how inadequate documentation affect the incentive or motivation of open-source newcomers. It also intends to serve as a starting point for further research on identifying potential strategies for improving documentation in order to increase newcomers' retention in open-source projects. Therefore, the following research question will be addressed:

*RQ: How does open source software documentation influence newcomers motivation to continue contributing to the community?*

We consider this an important question as open source software is widely used and relies on the contributions of a diverse community of developers. Understanding the factors that influence newcomers' motivation to contribute can help open source projects better support and retain new members, which is essential for the health and sustainability of the community [Hannebauer and Gruhn, 2017].

In order to get answers to our research questions, we will begin by providing an overview of the literature on the state of open-source documentation and the barriers that newcomers encounter. To identify pieces of literature, the snowballing search process will be used. We will then present our findings on these related works.

# Related work 3

## 3.1 Snowballing

To identify relevant literature we selected the snowballing search strategy. It is a research method in which an initial set of relevant articles is used to identify additional relevant articles, which are then added to the initial set [Wohlin, 2014]. This approach can be useful when the topic being researched is relatively new or when there is a lack of existing research on the topic. By starting with a small set of relevant articles and expanding upon it iteratively, the snowball literature search can help us identifying a broader range of relevant literature and gain a more comprehensive understanding of the topic at hand [Greenhalgh and Peacock, 2005].

After gathering a starting set of thematically relevant papers by using keyword search, we conducted the snowballing search in forward and backward directions.

As part of the backward iteration, we identified papers based on references from papers already in our starting set, while during the forward iteration we gathered relevant articles based on a reverse search looking for articles citing papers already in our starting set. We then repeated these iterations until no further papers could be found and added to the collection [Wohlin, 2014].

We extended and modified the snowballing search technique at two distinct places: (1) while gathering papers to be potentially included in the initial literature serving as the starting set, (2) and throughout the snowball research process any time when we would add or remove papers from our sets.

Regarding literature in fast-moving scientific fields, it is generally recommended to not include or consider references to articles or publications that are more than approximately 10 years old. While we agree with the overall sentiment for software topics in general, we also feel that it may be too restrictive in reference to merely using them to find additional potentially relevant papers. Therefore we relaxed this rule in such a way such as to tentatively allow older papers to be included in our starting and non-final working set in hopes of them helping us find and identify further references through forward snowballing iterations.

Second, we did both the initial search process as well as the iterations afterwards individually followed by a "cross-check" phase. This cross-check phase looked like the following:

1. If we both identified a paper as a valuable and relevant source then we instantly added it

to our working set.

2. If it only appeared in one of our sets, then we discussed it trying to determine its relevance - hopefully reaching a consensus whether to accept or discard it.
3. If such a state could not be reached then we included that tentatively in our set as well until such a time we can make a final decision on it.

## 3.2 Snowball search process

To imitate the literature review using the snowball approach, we first needed to identify a starting set that would serve as the baseline for the actual iterations. This starting set should be both relevant in its contents, published within the last approximately ten years, and preferably have a sufficient number of promising references as well as citations.

To identify the starting set, we identified primary keywords related to our research question [2.2] and compiled a list of synonyms for each to expand the scope of the search while maintaining the original intent and potentially finding more relevant materials.

Table 3.1 presents the primary keywords on the left with their respective synonyms appearing on the right row. Additionally, we also identified supplementary keywords, such as "retention", "availability", "quality", "quantity" and "onboarding" that may prove useful.

| Primary keywords | List of synonyms |
|---|---|
| Open source software | free and open-source software, open source, open-source, OSS, FOSS, FLOSS |
| newcomer | newcomers, beginner, novice, new contributor, first timer, first time contributor |
| documentation | software documentation, specification, code documentation, docs, document, documents, readme, information, comments, code comments, |
| barrier | fail, failure, hurdle, obstacle |
| retention | attrition |

***Table 3.1.*** Keywords and synonyms

To identify relevant literature, we developed a search expression using Table 3.1 through a process of iterative refinement, regularly evaluating the expression to ensure that it continued to produce an adequate number of relevant results after being modified. The final search string is presented in Figure 3.1.

Using this search string in Google Scholar resulted in a list of approximately 13.800 results. Out of these, we looked into and examined the papers from the first 20 pages, or 200 hits.

```
("open source" OR "free and open-source software" OR "open source" OR
"open-source" OR "OSS" OR "FOSS" OR "FLOSS")
AND
("newcomer" OR "newcomers" OR "beginner" OR "novice" OR "new contributor" OR
"first timer" OR "first time contributor")
AND
("docs" OR "documentation" OR "software documentation" OR "code documentation" OR
"document" OR "documents" OR "readme" OR "information" OR "comments" OR
"code comments")
AND
("retention" OR "barrier" OR "fail" OR
"failure" OR "hurdle" OR "obstacle" OR "attrition" OR "availability" OR
"quality" OR "quantity" OR "onboarding")
```

***Figure 3.1.*** Search string

## 3.3 starting set

We went through the hits of the Google Scholar keyword search by the method of what we previously defined as "individual then cross-check" that we used extensively during every phase of the snowballing search process.

In analyzing the presented results, we followed a systematic review process, starting with an examination of the titles and then proceeding to the abstracts. We then moved on to the introduction, conclusion, and findings sections, and if we were still unable to determine the relevance of the paper, we performed a full text read. We applied relaxed time constraints, allowing relevant papers to be considered from outside the typical range of the past 10 years or so to be considered if they were deemed to be a valuable source of forward snowballing references.

The final starting set can be seen in Table 3.2.

| Date | Title of paper |
|---|---|
| 2022 | "Understanding community participation and engagement in open source software Projects: A systematic mapping study" [Kaur et al., 2022] |
| 2014 | "The hard life of open source software project newcomers" [Steinmacher et al., 2014c] |
| 2014 | "Barriers Faced by Newcomers to Open Source Projects: A Systematic Review" [Steinmacher et al., 2014b] |
| 2015 | "Supporting newcomers to overcome the barriers to contribute to open source software projects" [Steinmacher, 2015] |
| 2015 | "A systematic literature review on the barriers faced by newcomers to open source software projects" [Steinmacher et al., 2015] |
| 2014 | "DMOSS: Open source software documentation assessment" [Carvalho et al., 2014] |
| 2019 | "Software Documentation Issues Unveiled" [Aghajani et al., 2019] |
| 2021 | "Understanding Emotions of Developer Community Towards Software Documentation" [Venigalla and Chimalakonda, 2021] |
| 2001 | "Open-source documentation: in search of user-driven, just-in-time writing" [Berglund and Priestley, 2001] |
| 2009 | "Measuring Open Source Documentation Availability" [Matuleviçius et al., 2009] |
| 2010 | "Creating and evolving developer documentation: understanding the decisions of open source contributors" [Dagenais and Robillard, 2010] |
| 2017 | "Evaluating the Quality of the Documentation of Open Source Software" [Aversano et al., 2017] |
| 2013 | "Why do newcomers abandon open source software projects?" [Steinmacher et al., 2013] |
| 2018 | "Newcomers' Barriers. . . Is That All? An Analysis of Mentors' and Newcomers' Barriers in OSS Projects" [Balali et al., 2018] |
| 2014 | "How to Support Newcomers Onboarding to Open Source Software Projects" [Steinmacher and Gerosa, 2014] |
| 2018 | "Understanding Newcomers Success in Open Source Community" [Bayati, 2018] |
| 2015 | "Supporting newcomers in software development projects" [Panichella, 2015] |
| 2017 | "Why modern open source projects fail" [Coelho and Valente, 2017] |

***Table 3.2.*** Starting set

## 3.4 Iterations

After finalizing the starting set, we conducted the snowballing iterations. Similarly to the process of assembling the starting set, we performed the iterations individually and cross-checked each other's work. Each iteration consisted of a backward and a forward search. During the backward snowball search we examined the papers and tried to identify relevant articles in the references section of a paper going through them with the by reading the title, abstract, introduction, conclusion and findings followed by a full read if needed. For the forward snowball search we applied the selfsame selection principles but searched for the papers from our list in Google Scholar and attempted to find relevant ones through a reverse search showing the list of articles

citing one of the previously included papers.

### 3.4.1 First Iteration

Over the course of the first backward iteration, from the 18 papers of the starter set, we gathered 33 additional papers to be tentatively included in our final set based on their titles after excluding duplicate founds (meaning the same papers being referenced to from multiple items of our starter set) as well as filtering them by their creation dates to only get papers from the last 10 years. The references were tightly interwoven, with half of the tentatively included set coming from the first three papers inspected after removing them from the lists of all but one - meaning if we were to redo the backward iteration in arbitrary order, the last few papers would barely have additional candidates.

After having gathered the promising references, we reviewed then discussed them weeding out unrelated papers. In the end, 8 of the original 18 tentative papers were actually included in the baseline set for the next iteration.

During the first forward iteration, we collected another 13 tentative papers after deduplication and applying date constraints. After reviewing their contents, we included a total of 5 additional papers in the next iteration.

Lastly, we went through the papers of the starting set one more time and removed those that we had only included as potential sources of forward references.

### 3.4.2 Second Iteration

This way we began the second and final iteration using the papers identified during the first one in search of references. This round proved to be much less fruitful, with merely 3 backward and 3 forward candidates identified. After reviewing them, none were added to our final list. Therefore, the iterative process of identifying relevant literature concluded and we have arrived at the final set.

## 3.5 Final set

Our final set of related works consists of the 18 papers showcased in Table 3.3. While all these papers identified contain relevant parts related to our research question [2.2], none of them is an exact match, and none of them focuses on the same topic as we do, indicating the need for further research in this area.

| Date | Title of paper |
|---|---|
| 2014 | "The hard life of open source software project newcomers" [Steinmacher et al., 2014c] |
| 2015 | "Supporting newcomers to overcome the barriers to contribute to open source software projects" [Steinmacher, 2015] |
| 2015 | "A systematic literature review on the barriers faced by newcomers to open source software projects" [Steinmacher et al., 2015] |
| 2019 | "Software Documentation Issues Unveiled" [Aghajani et al., 2019] |
| 2017 | "Why modern open source projects fail" [Coelho and Valente, 2017] |
| 2017 | "Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey" [Lee et al., 2017] |
| 2018 | "Determining Newcomers Barrier in Software Development: An IT Industry Based Investigation" [Showkat, 2018] |
| 2014 | "Preliminary Empirical Identification of Barriers Faced by Newcomers to Open Source Software Projects" [Steinmacher et al., 2014a] |
| 2019 | "Ten simple rules for helping newcomers become contributors to open projects" [Sholler et al., 2019] |
| 2017 | "Difficulties of Newcomers Joining Software Projects Already in Execution" [Matturro et al., 2017] |
| 2014 | "Older Adults and Free/Open Source Software: A Diary Study of First-Time Contributors" [Davidson et al., 2014] |
| 2017 | "Exploring Knowledge Loss in Open Source Software (OSS) Projects" [Rashid et al., 2017] |
| 2022 | "Managing Episodic Volunteers in Free/Libre/Open Source Software Communities" [Barcomb et al., 2020] |
| 2011 | "A field study of API learning obstacles" [Robillard and DeLine, 2011] |
| 2015 | "How API Documentation Fails" [Uddin and Robillard, 2015] |
| 2016 | "Understanding the Factors That Impact the Popularity of GitHub Repositories" [Borges et al., 2016] |
| 2014 | "Co-evolution of project documentation and popularity within github" [Aggarwal et al., 2014] |
| 2018 | "Open source barriers to entry, revisited: a sociotechnical perspective" [Mendez et al., 2018] |

***Table 3.3.*** Final set papers

## 3.6 Related work

There has been numerous ongoing studies using various research methods on the topic of open-source software including but not limited to documentation issues as well as works identifying frequently occurring barriers newcomers face when trying to join an open-source project.

The general agreement is that there are four big types of barriers present hindering newcomers' progress. In this section, we highlight a few of those most commonly occurring ones in open-source projects in addition to going into a little bit more detail trying to understand why they occur, how they are relevant to our project and what can be done to counteract the negative effects of them.

### 3.6.1 Non existing

Perhaps the biggest issue of all is if the given project does not have any kind of documentation artifacts like mainly general description of the software and architecture, a documentation landing page, READMEs, setup guides, licenses and others [Coelho and Valente, 2017]. Software projects can be incredibly large and of complex structures - understanding them just from the code would be an immense overtaking that not everyone has the time or willingness for, especially if there are alternatives with it and as is the nature of open-source, no monetary compensation is present either.

Numerous authors in the literature over the last decade have identified a lack of documentation as a barrier to newcomers. More so than a single barrier among others, non-existing documentation is thought to be the biggest problem out of all issues related to documentation as studies point it out [Showkat, 2018].

Coelho and Valente [Coelho and Valente, 2017] found that the absence of documentation pieces strongly correlate with the project being eventually abandoned. Steinmacher et al. [Steinmacher et al., 2014a] argued that deficiency in terms of documentations describing the overall working of the code and the project especially hurts newcomers that are also relatively new to software development in general, like undergraduates or recent graduates. It is easy to see why, senior or lead developers with decades of experience knowing general patterns like the back of their hands can a lot more easily identify key parts of the code guessing what and how they do then their junior counterparts. At a later time in another research of his, Steinmacher [Steinmacher, 2015] further confirmed these findings through various research methods including interviews with contributors and graduate would-be contributors.

Matturro et al. [Matturro et al., 2017] researched the topic of difficulties newcomers encounter when joining software projects already in execution. They too identified the lack of planning and documenting as a hurdle to be overcome as more than one third of their respondents cited the aforementioned two points.

### 3.6.2 Outdated

It has been demonstrated through various studies that the presence of documentation in an open-source project does not necessarily guarantee the usefulness or relevance of its contents. We consider a document out-of-date if it is not in sync with the systems it attempts to describe. Aghajani et al. [Aghajani et al., 2019] also investigated software documentation issues. They found that up-to-dateness problems account for almost forty percent of all documentation issues in software. In some ways, outdated documentation can be tantamount to a complete absence of documentation if the majority of the code has since undergone significant revisions since its inception. Often documentation is made by a few or even a single contributor, which then can quickly become outdated as that person stops contributing to the software, or starts focusing

on other parts of the project instead.

Steinmacher [Steinmacher, 2015] found that finding documentation that is no longer relevant can quickly lead to demotivation. Naturally, they found older repositories to be more prone to this kind of hindrances; which also doubles down as older projects tends to also be larger making the effective lack of proper documentation even more distressing for newcomers. They suggest that at the very least, marking documents as outdated should be done to prevent newcomers from wasting time while also setting their expectations [Steinmacher, 2015], [Sholler et al., 2019].

Steinmacher et al. [Steinmacher et al., 2015] further argues the importance of up to date documentation stating that as even if there exist some documentation artifacts, if they are outdated they can prove to be barriers impeding knowledge rather than being a useful tool for the developers [Uddin and Robillard, 2015], [Showkat, 2018]. Related this issues are also cases where the users simply do not have any meaningful ways to determine the creation and update time, therefore the validity of doc fragments that they find. Conversely, overzealous attempts at fixing the aforementioned issues can lead to a situation where there is way too much and overly verbose information available leading to information overload for newcomers. In spite of this all, Forward and Lethbridge [Forward and Lethbridge, 2002] argues that developers simply learn how to deal with it lessening the importance of this particular obstacle.

### 3.6.3 Unclear

The concept of clear documentation encompasses various subtopics related to the content of software documentation and has been explored in literature on documentation issues. Quality documentation is necessary for developers to understand the codebase, especially so during the entry into a project [Lee et al., 2017]. In "A Field study for API learning obstacles" Robillard et al. [Robillard and DeLine, 2011] mention examples from participants like "it was not clear how to instantiate an object, there were too many abstract classes, the names did not make sense" and "If it's not clear how I match APIs with their scenarios, if I need to draw a circle on the screen, and I don't see something that clearly says, 'this is how you draw', I will say that's complex".

In the study made by Robillard et al. [Robillard and DeLine, 2011] content-wise incompleteness and ambiguity were the top issues. Both of these issues can occur in either in the form of in-code comments or in standalone documents. According to Steinmacher et al., [Steinmacher et al., 2015] unclear documentation can hinder the understanding of the codebase, rather than serving as a helpful resource, much like the case of outdated documentation. By definition a work that is incomplete lacks all the necessary or appropriate parts and is therefore not finished. Incomplete documentation can take on many forms: there are cases, where simply part of the software documentation are missing, many times major parts of a documentation are auto-generated from inline comments, using trivial information that can be gathered by a

software parsing the code like the basic signature of a method consisting of it's name, return type, and the possible parameters it can take. However, the more complex a functionality a method has, the more developers would need clear documentation on do's and don'ts, what side effects and possible limitations it may have, how and where is it used. Ambiguity is another form of incompleteness where the missing piece of the puzzle is the most important one: documentation leaving out in depth explanation of key parts therefore making different explanations possible for a piece of functionality making readers confused about the purpose of a class or method. For the latter how and where a piece of code can be used, the most common issue found was missing examples and inadequate explanation of an example [Robillard and DeLine, 2011].

Steinmacher et al. [Steinmacher et al., 2014c] [Steinmacher et al., 2014a] as well as Mendez et al. [Mendez et al., 2018] also found examples not being present in addition to unclear terminology and terms and abbreviations not being explained as issues, but also that in general these types of issues are more prevalent among those who are in the early stages of joining a project, newcomers. Perhaps they come with experiencing in navigating the files and source of a new software project.

### 3.6.4 Fragmented or disjointed

All the following issues are related to the documents related to a piece of software being inadequately presented [Aghajani et al., 2019]. This can surface in the forms of like documentational artifacts being fragmented between multiple places or although being in the same place having a bad flow making it a bigger effort to understand than it would be necessary. In their mixed-approach multi-phased study about API learning obstacles, Robillard et al. [Robillard and DeLine, 2011] found that many participants thought existing documentation can also severely lack in the department of format and quality. They mentioned boilerplate documentation as well with one-liner explanations that are usually just the API endpoint names rephrased as well as overly trivial examples with a single or two method calls. They concluded that developers vastly prefer a relatively continuous, long, single document to a lot of smaller snippets at various places. Furthermore, they cited that the documents being fragmented makes the information less discoverable [Robillard and DeLine, 2011]. Having arrived to similar results in their respective qualitative studies, Steinmacher et al. [Steinmacher et al., 2014c] and Showkat [Showkat, 2018] mentioned spread documentation being a barrier as well. Lastly, Sholler et al. [Sholler et al., 2019] extends this concept by suggesting putting documentations in few but easy-to-find places, as newcomers getting lost between them is a real concern.

Robiliard et al.'s [Robillard and DeLine, 2011] participants were also on a consensus regarding the flow of the documentation. It should follow the flow of the computer program, preferably continuously instead of having to open ten to twenty additional pages to get an overarching idea of a single call. They also expanded upon what is considered a proper, nice flow. The majority of the participants in their study concurred that it can be equally overwhelming to be presented with a large amount of raw, overly formal and precise documentation, as it can be

disorienting to encounter poorly structured and segmented documentation.What can alleviate this concerns they found is the documentation having a story-like flow from its beginning to the end expanding upon the concepts in detail it encounters while going through the whole program flow and structure at the same time. Though they focused on general API learning obstacles, their findings can also be applied in the case of open-source software learning barriers.

These papers all mentioned parts related to our research, as they discussed a fair amount of presentation related issues in software documentation, however what they fail to mention is why these issues are present in the first place together with how do these documentation issues among others actually affect newcomer retention in an open-source project.

### 3.6.5 Further issues

Additionally, there are several other important considerations to be noted in the context of documentation in open-source software development. For starters, documentation quality in regards to its contents can be wildly inconsistent [Robillard and DeLine, 2011]. The lack of contributing guidelines [Davidson et al., 2014], READMEs, licenses, workplace setup guidelines [Davidson et al., 2014], [Steinmacher et al., 2014a], [Coelho and Valente, 2017], are also mentioned by multiple authors just as an overwhelming amount of documentation [Steinmacher, 2015] as a stark contrast to lack of documentation already mentioned above.

However, even if the documentation is present, it has a nice flow, has just the right size and presentation as well it can still has issues like ambiguity, contradicting information and incompleteness, that contributors only realise once they encounter problems trying to rely on that [Uddin and Robillard, 2015], [Aghajani et al., 2019].

Another documentation-like artifact is the topic of code comments mentioned by various authors. They can be a good resource too to understand classes and methods, but just as in the case of outdated or ambiguous documentation among others, if they are not clear enough, they can prove to be more of a hindrance than helpful resource as well [Steinmacher et al., 2015], [Showkat, 2018].

Lastly, participants of Sholler et al.'s [Sholler et al., 2019] study mentioned a lack of "how to contribute" walkthroughs for newcomers as well. This would be able to also alleviate fears of newcomers on whether to contribute to the project or not.

Issues like all the aforementioned ones have been correlated in general eventual abandonment of software projects [Coelho and Valente, 2017].

# Methods 4

## 4.1 Case studies

As for our research methodology, we settled on using case studies. Through accurate and comprehensive study of our select real-world cases we aim to acquire a more in-depth and thorough understanding of a complex problem that is the influence of the quality and existence of open-source software documentation on newcomers to the software project.

We extended case studies with the inclusion of a negative case. By using both a positive and a negative case in our study serving as a base of comparisons between them we further enhanced the validity of our qualitative research resulting in a more objective and probably less biased result than without while also serving as to understand and explain the original case better [Emigh, 1997] [Hanson, 2017]. A base of comparison serves as a showcase presenting the difference between the outcomes occurring predicted by the theory if certain conditions have been met (in our case, abundant and helpful documentation artifacts helping newcomers onboard an open-source project) and when they are not [Emigh, 1997].

The subjects of our case study are two open-source projects: FlyByWire Simulations A32NX and VLC media player.

## 4.2 Case descriptions

### A32NX

The A32NX Project is an open-source project developing a high-quality, fully functional, and free A320neo aircraft for Microsoft Flight Simulator 2020. Their aim is to make it as close as possible to the real aircraft with the same name produced by Airbus SE. Initially starting out at 2020 august as an enhancement to the default A320neo shipped with the game, more than two years later it is now an independent add-on project maintained and developed by the FlyByWire Simulations community of more than 200 individual contributors.

During its 2 years of active development, the A32NX project has grown to be one of the most popular extensions of the simulator, receiving several thousand individual modifications and add-ins. Furthermore, being hosted on GitHub, the main aircraft project has amassed 4505 stars, 888 forks, and 3839 commits in addition to 2071 approved and merged pull requests

Besides the main repository, since then the community spawned various related sub-projects shared by the FlyByWire Simulations umbrella including one separate project for the documentation, one for integration with third-party data sources and services, and one for the installer and updater functionalities.

The community as of today (2022.12.13) is in very good health. The project has developer as well as thorough end-user documentation. It also has clear contribution guidelines, including a code of conduct, an introduction to their version control and pull request systems, review and QA processes, help finding tasks and issues to work on, and development setup guides. For additional questions and quick discussions, the community uses the Discord instant messaging social platform. The average response times can be considered quick; developers who have a question or are stuck with a problem can expect answers in minutes to hours.

On average, on GitHub alone, their issue tracker sees 4-5 reports in addition to 10-15 pull requests being made by contributors each week.

#### VLC media player

VLC media player is a free and open-source multimedia player. One of its main advantages is its ability to play a wide range of media formats  without the need for additional codecs or plugins. It is available on all mayor platforms, including Windows, macOS, Linux, Android, and iOS, and can be used to play media from local storage or streamed over the internet.

In addition to its basic media playback capabilities, VLC media player also has a number of advanced features that make it a powerful tool for a variety of purposes. For example, it supports streaming protocols, allowing users to play media from a variety of sources, including online video and audio streams.

The VLC media player also offers a high level of customization, with a number of options for adjusting the appearance and behavior of the player. Users can choose from a variety of skins to change the look of the player, and there are also options for adjusting the audio and video settings, as well as for enabling or disabling various features.

As of today (2023.01.02), the VideoLAN Client (VLC) media player has been in active development for almost 22 years. The development community is very active and is still seeing multiple commits every day on their self-hosted GitLab repository.

It is clear, that contributors or the VLC media player were once maintaining documentation, as we can find various fragments of it scattered throughout their domain.

Unfortunately, by now, the documentation for the Video LAN project is widely dispersed and frequently outdated, with multiple dead links and disconnected documentation systems. On a positive note, there is separate documentation for users and developers, with the latter having

more issues, which is more unfortunate for would-be contributors. The overall documentation is inadequate, as indicated even by a disclaimer on the project's website stating that they are unable to provide up-to-date documentation until further notice due to a lack of volunteer technical writers. Counterintuitively, the simpler concepts often receive more detailed coverage and overly verbose explanations, while more advanced topics are often lacking or outdated.

Overall, the documentation for this project is highly disorganized and difficult to navigate due to its fragmentation and outdated nature. It is scattered across multiple locations and not clearly linked together, making it challenging for users to find the information they need. This, in turn, can be frustrating for users who are trying to learn about the project or its features.

## 4.3 Data collection

Qualitative data collection entails obtaining non-numerical data, such as opinions, attitudes, behaviours, beliefs, and values. Considered qualitative data collection methods include interviews, observations, focus groups, surveys and repository mining techniques. In order to get less biased interpretations of our cases, out of those considered, we chose two collection methods: interviews and observations.

### 4.3.1 Interview

Interviews are a widely used method of collecting qualitative data by asking respondents questions and having them provide verbal responses. In the case of interviews, the interviewer is in direct (but not necessarily physical) contact with the interviewees while collecting data in real time. Interviews can be conducted in-person, over the phone, or online over video chat.As participants of open source projects projects like FBW A32NX and VLC are not limited to certain countries or regions as is the case with many enterprises. Considering this, physical interviews are not feasible, and conducting group interviews or focus group interviews would prove to be difficult as well due to the inherent problem of different locations and time-zones. Therefore, we settled on using online video interviews with individual participants.

We are planning on conducting a series of "semi-structured interviews" popular in case studies, containing both structured and unstructured elements [Runeson and Höst, 2009]. That means that the questions are open-ended, allowing the respondent to answer in their own words, rather than providing them with a list of predetermined responses. Also, while we have a list of questions to be asked, they are not strictly asked in order, and the respondent would also be allowed to provide additional information or to elaborate on their answers [Runeson and Höst, 2009].

During the interviews, an audio recording would be made as well, further allowing us to gather even more details by compiling the recording into a written transcript as well.

To that end, we have gathered a preliminary list of questions below:

**Interview guide**

Introductory questions

- Can you tell a bit about your professional experiences?
- How many years of experience do you have as a software developer?
- What is the highest level of education that you have finished?
- Have you contributed to open-source projects before, and if so, which ones?

Newcomerness questions

- What motivated you to start contributing to this open-source project?
- How did you learn about the project and what steps did you take to get started?
- Can you describe your initial impressions of the project when you first started or tried to start contributing?
- How did you start your interactions with the community? (mailing list, IRC chats, forums, PRs)? *(if applicable)*
- Does the project has an onboarding process, are there documents to help onboarding?
  - If yes, did you find it helpful or valuable?
  - Were there any challenges or barriers that you encountered during the onboarding process, and if so, how were they resolved?
  - Can you take me through the onboarding process and how it helped you to contribute?

Contribution / documentation problems questions

- Did you face problems trying to contribute?
- Have you successfully submitted a PR? How was the experiences and the challenges?
- How easy or difficult was it to find the information you needed to start contributing?
- Does the project has documentation? Did you find written information helping you?
- How did you find the documentation available in the community?
- Has anyone from the community helped to solve a problem you have faced? From what source did you get help?
- Did you consult the documentation as part of you trying to contribute? Have you looked into it?
  - Did you find the answers you sought in the docs?
  - How much time did you spend looking at docs?
  - Is there any part of it that is lacking? Contribution guidelines, user documentation, developer documentation, how-to guides, READMEs?
  - In what way is it lacking? Too much, too little, out of date, etc.
  - How severe were the documentation problems you encountered?
  - How frequently (if ever) did you encounter problems with the documentation?

- Can you show us an example of documentation and tell how it helped you to contribute?
- Do you have any specific ideas for improving the documentation that could help newcomers out?
- Do you even think it is needed?

General thoughts and feelings

- What other difficulties did you face while trying to contribute to this open-source projects?
- What do you think other newcomers find the most difficult if they want to start contributing to this project?
- Are there anything that you thought would be completely different?
  - What surprised you?
  - What did you like?
  - What did you not like about working on this project?
- All in all, was your experience with the project positive?
- Would you continue contributing to this specific project?

### 4.3.2 Observation

This technique involves the researcher monitoring a person or a group of people as they are going about their activities. This is an effective way of collecting qualitative data as it can provide an insight into how people act and interact in their natural environment. Observation is an indirect data collection method - we collect data in real time without interacting with the subjects [Lethbridge et al., 2005]. It gives the opportunity to observe behaviors or events that may be difficult to capture through verbal communication. We supplement the information gathered through interviews with observations of both positive and negative cases to capture more detail than we would be able to with interviews alone [Espedal et al., 2022]. If we find discrepancies arising between the two data sources, further investigation may be needed. Precisely because interaction is eliminated from the process, observations can provide an objective and unbiased data source. Interviews can be subject to the interviewer's bias or influence, but observations allow for a more impartial collection of data. They remove a little bit of the subjective, often subconscious human factor, the inability or unwillingness to disclose specific details or notes of personal failures [Lethbridge et al., 2005] [Espedal et al., 2022]. In this case, we will be conducting online observations of a specific open-source contributor or would-be contributor, including their interactions with the community, preferably across different platforms for the selected project. The focus of our observations will be the online activities of the newcomer, trying to follow and trace them as well as utilize field notes to document their participation.

Putting it in practice, the difficulty with regard to observations is the selection of a specific subject to observe.

To that end, there are certain considerations to be made, gathering various criteria that have to be met both for cases serving as positive and negative examples. What makes a subject and their activities relevant:

1. The person whose activity is being observed is a newcomer. We are starting with Steinmacher's ([Steinmacher, 2015]) definition here of who a newcomer is: People are actively trying to make their first contribution to an open-source project. The first keyword here is "first contribution". They have only recently started to get involved with the community, lacking both technical knowledge, the inner workings of the software, and also might not be aware of certain conventions, approaches or ways of doing thing in a select project. The second is "actively trying" as in they are in the process of modifying parts of the software or documentation and trying to get their changes accepted. During this process, they might even interact with the community at their support or discussion forums, IRC, version control system, issue tracker, or mailing list to ask for help, get feedback, or discuss the changes.
2. Their interactions preferably should have occurred within the last 1-2 years. The case considered should be fairly recent, as older events might not necessarily showcase the current state of the community, documentation, newcomer resources, or the lack thereof.

Field notes of our findings would be taken during the observations. They will help us to accurately record the details of our observations, including the behavior and actions of the subjects being observed. This can be particularly important in case a specific observation takes longer or has a bigger scope than previously anticipated. Additionally, writing field notes will also help us develop a deeper understanding of the case, by helping us to pay close attention to the little details and reflect on the significance of what we are seeing, as well as being helpful when it comes time to analyze and interpret the data.

Potential further things to look out for:

- The quality and frequency of the newcomer's contributions indicate the level of engagement and commitment the newcomer has to the project.
- The level of communication with other developers.
- The amount of support and guidance the newcomer seeks from other members of the community.
- Involvement in community events and activities.

## 4.4 Data analysis

### 4.4.1 Thematic analysis

Thematic analysis is a qualitative data analysis method that is both accessible and flexible as it can be conducted in different ways. It provides a way to systematically identify, organize

and explain key concepts of select themes across the analysed data set. As thematic analysis is focuses on the meaning of themes in parts of the data set, it allows us to connect the dots; to relate the stories of separate open-source contributors and find commonalities between them based on their circumstances and experiences. The final task and the overall outcome of the thematic analysis process is to identify the themes pertinent to answering our research question [Braun and Clarke, 2006].

As our research question is exploratory, the thematic analysis is experimental as well as inductive, as the data used is based on the interview and observation participant' experiences.

We will follow the six-phased thematic analysis approach developed by Braun and Clarke [Braun and Clarke, 2006]. It begins with immersion in the data set, followed by labeling (coding) of the data and then a shift from codes to themes that captures and represents some key concepts relevant to the research question. Finally, a detailed report of the findings can be made using clearly defined and named themes.

#### Phase 1: Familiarization

The first phase is to get immersed in the data and become intimately familiar with it. This is achieved through going by the data multiple times. This can be accomplished through multiple iterations of listening to audio or watching video recordings, reading and re-reading the transcripts over and over again. However, merely reading is inadequate by itself. One has to think about the meaning behind the words. It may also be helpful to make notes about relevant aspects related to the research question. This initial phase is crucial for gaining a comprehensive understanding of the data.

#### Phase 2: Generating initial codes

After getting thoroughly familiar with the data set, the next phase deals with labelling or coding it. Here we make note of parts of the data that can potentially be relevant to the research question. This code is typically a one to few word summary or description of a data point, but not at the level of a fully developed interpretation of it. Labels can be applied at a granular level, such as to individual lines or paragraphs, to provide a more precise level of coding for the data.

#### Phase 3: Searching for themes

With the pre-processing steps out of the way, we make a shift from raw data to coded data, and from coded data and codes to themes. The codes already included some level of interpretation in the previous phase, but more as a side product of creating summaries for labels. This is the phase where we really focus on the interpretation. A theme captures an important part of the data that is related to the research question and attaches some kind of explanation and

interpretation to it. Basically identify and explain the main topics of interest. During the second phase of thematic analysis, it is important to consider the connections and relationships between themes. This can be achieved by analyzing the data and creating a table that organizes the identified themes and displays relevant parts of the data set beneath them. This allows for a more comprehensive understanding of the data and can inform the interpretation of the results. It is essential to keep in mind that this phase involves more than just analyzing individual data points and requires a holistic perspective.

**Phase 4: Reviewing potential themes**

We can consider this a quality assurance phase. The themes gathered so far are compared against the data set, checking whether it supports their points. We should also make considerations regarding the properties of the themes. In order to ensure that the themes are of good quality, it is important to consider several factors, including whether they are well-defined with clear boundaries and a singular focus, whether they are based on a sufficient amount of data points, and whether they directly address one or more aspects of the research question. By carefully evaluating the themes in this manner, it is possible to ensure that they are of high quality and can be used to effectively answer the research question.

**Phase 5: Defining and naming themes**

Following the previous quality check phase, we are improving upon the themes we settled on using. Themes are specified more precisely, going into finer details, and are also further developed in relation to the other themes. This is also the place where we interpret the themes' data, draw conclusions from it, and start preparing the overarching narrative of the story, including how and when to include the themes and how to connect them together.

**Phase 6: Producing the report**

Lastly, a detailed report has to be made in the form of a coherent and compelling story telling our findings in a clear and persuasive manner.

## 4.5 Validity

In this section, we will discuss and address the various threats to validity in a systematic way that may have influenced the results of the study, and describe the steps we will take to minimize these threats. This aids our readers in assessing the trustworthiness of the research and determining whether the conclusions are supported by the data and helps increasing the trustworthiness of the research findings so it they do not reflect subjective views or biases [Lethbridge et al., 2005] [Runeson and Höst, 2009].

Here, we opted to classify and address them the way suggested and used by Yin [Yin, 2003] and Wohlin et al. [Wohlin et al., 2012] The four areas of validity in question are as follows:

### 4.5.1 Construct validity

Construct validity refers to the extent to which our research accurately assesses the specific theoretical concept that it is intended to measure. In other words, how well does the research represent what we had in mind and what the research question aims to investigate, what it measures, and what it is supposed to be measuring [Runeson and Höst, 2009]. This type of validity helps to ensure that the results of a study are meaningful and interpretable.

To ensure construct validity in our study, we will take the following steps:

1. Clearly define the concept we are studying and identify the specific aspects of it that our research question aims to investigate. This will ensure that our study design is focused on accurately measuring the concept of interest.
2. Select appropriate measures and methods for collecting data on the concept being studied helps ensuring that the data we collect accurately reflects the concept we are trying to measure.
3. Interpret the results carefully, taking into account the limitations of our measures and the study design. This will help to ensure that the conclusions we draw from the data are valid and meaningful.

### 4.5.2 Internal validity

When examining causal relations, it is vital to investigate internal validity. It is a way to measure how confident we can be that the results of the study are true and show a causal relationship (cause and effect) between variables observed, dismissing the notion that the effect happened due to some unobserved external factors. To that end, we have to be aware of external variables and how they might affect the investigated one.

To mitigate threats to internal validity, we are using multiple research methods combined with multiple cases, both a positive and a negative one as well strengthening our case more. Performing multiple interviews and observations also results in a larger sample size, further reducing the influence of extraneous variables as well as increasing the power of the study making the repeatedly observed effects statistically more significant.

### 4.5.3 External validity

It is the extent to which the results or conclusions of our study can be generalized or applied to other situations, cases beyond the conditions in which the study was conducted.

The findings of this study are about select two communities. However, to strengthen our research, we are using multiple research methods (interviews and observations) and interacting and gathering information from multiple participants in both cases. Furthermore, many open-source communities have similar characteristics. Open-source communities are generally focused on a specific project or set of projects and operate using a decentralized development model, with contributors working together to develop or improve software while also emphasizing collaboration and knowledge sharing. Thus we believe that our findings would likely be applicable to other, similar open-source communities as well.

### 4.5.4 Reliability

Reliability or dependability is a measure of the consistency and repeatability of the study. It answers the question: how precisely would it be reproducible by other researchers, and would they also arrive to the same conclusions? So how dependent are the findings of the research on the subjective views and biases of the researchers producing them.

To ensure the reliability of our findings, we thoroughly document all parts of the research process. What research methods did we settle on using, as well as the justification of it and the various decisions made throughout the procedure. How, and based on what criteria, did we find the cases and participants. What data analysis tools did we use to arrive at our interpretation from the raw data.

# Conclusion 5

## 5.1 Conclusion

In conclusion, open-source software plays a significant role in the modern technological landscape and has the potential to facilitate knowledge transfer within communities of developers. However, barriers exist that can prevent newcomers from effectively participating in open-source projects. A subset of these issues are related to documentation: be it too little, too much, or simply outdated, neither of them are as helpful as we would want them to be.

To better understand these barriers and how they can be overcome, we have planned a research study for next semester that will use a case study and qualitative data analysis techniques to explore these issues. The thematic analysis of the data collected in this study will provide valuable insights into the experiences of newcomers in open-source projects, particularly in answering the question of how open source software documentation influences newcomers' motivation to continue contributing to the community.

# Bibliography

[Aggarwal et al., 2014] Aggarwal, K., Hindle, A., and Stroulia, E. (2014). Co-evolution of project documentation and popularity within github. In *Proceedings of the 11th working conference on mining software repositories*, pages 360–363.

[Aghajani et al., 2019] Aghajani, E., Nagy, C., Vega-Márquez, O. L., Linares-Vásquez, M., Moreno, L., Bavota, G., and Lanza, M. (2019). Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1199–1210.

[Aversano et al., 2017] Aversano, L., Guardabascio, D., and Tortorella, M. (2017). Evaluating the quality of the documentation of open source software. In *ENASE*, pages 308–313.

[Balali et al., 2018] Balali, S., Steinmacher, I., Annamalai, U., Sarma, A., and Gerosa, M. A. (2018). Newcomers' barriers... is that all? an analysis of mentors' and newcomers' barriers in oss projects. *Computer Supported Cooperative Work (CSCW)*, 27(3):679–714.

[Barcomb et al., 2020] Barcomb, A., Stol, K.-J., Fitzgerald, B., and Riehle, D. (2020). Managing episodic volunteers in free/libre/open source software communities. *IEEE Transactions on Software Engineering*.

[Bayati, 2018] Bayati, S. (2018). Poster: Understanding newcomers success in open source community. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pages 224–225. IEEE.

[Berglund and Priestley, 2001] Berglund, E. and Priestley, M. (2001). Open-source documentation: In search of user-driven, just-in-time writing. In *Proceedings of the 19th Annual International Conference on Computer Documentation*, SIGDOC '01, page 132–141, New York, NY, USA. Association for Computing Machinery.

[Borges et al., 2016] Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the factors that impact the popularity of github repositories. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*, pages 334–344. IEEE.

[Braun and Clarke, 2006] Braun, V. and Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3:77–101.

[Carvalho et al., 2014] Carvalho, N., Simões, A., and Almeida, J. (2014). Dmoss: Open source software documentation assessment. *Computer Science and Information Systems*, 11:1197–1207.

[Coelho and Valente, 2017] Coelho, J. and Valente, M. T. (2017). Why modern open source projects fail. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, pages 186–196.

[Dagenais and Robillard, 2010] Dagenais, B. and Robillard, M. P. (2010). Creating and evolving developer documentation: understanding the decisions of open source contributors. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 127–136.

[Davidson et al., 2014] Davidson, J. L., Mannan, U. A., Naik, R., Dua, I., and Jensen, C. (2014). Older adults and free/open source software: A diary study of first-time contributors. In *Proceedings of the international symposium on open collaboration*, pages 1–10.

[Emigh, 1997] Emigh, R. J. (1997). The power of negative thinking: The use of negative case methodology in the development of sociological theory. *Theory and Society*, 26:649–684.

[Espedal et al., 2022] Espedal, G., Løvaas, B., Sirris, S., and Wæraas, A. (2022). *Researching Values. Methodological Approaches for Understanding Values Work in Organisations and Leadership*.

[Forward and Lethbridge, 2002] Forward, A. and Lethbridge, T. C. (2002). The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering*, DocEng '02, page 26–33, New York, NY, USA. Association for Computing Machinery.

[GNU, 2022] GNU (2022). What is free software?

[Greenhalgh and Peacock, 2005] Greenhalgh, T. and Peacock, R. (2005). Effectiveness and efficiency of search methods in systematic reviews of complex evidence: audit of primary sources. *BMJ*, 331(7524):1064–1065.

[Hannebauer and Gruhn, 2017] Hannebauer, C. and Gruhn, V. (2017). On the relationship between newcomer motivations and contribution barriers in open source projects. In *Proceedings of the 13th International Symposium on Open Collaboration*, OpenSym '17, New York, NY, USA. Association for Computing Machinery.

[Hanson, 2017] Hanson, A. (2017). *Negative Case Analysis*, pages 1–2. John Wiley Sons, Ltd.

[Kaur et al., 2022] Kaur, R., Kaur Chahal, K., and Saini, M. (2022). Understanding community participation and engagement in open source software projects: A systematic mapping study. *J. King Saud Univ. Comput. Inf. Sci.*, 34(7):4607–4625.

[Lee et al., 2017] Lee, A., Carver, J. C., and Bosu, A. (2017). Understanding the impressions, motivations, and barriers of one time code contributors to floss projects: a survey. In *2017*

*IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 187–197. IEEE.

[Lethbridge et al., 2005] Lethbridge, T., Sim, S., and Singer, J. (2005). Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, 10:311–341.

[Li et al., 2015] Li, Z., Avgeriou, P., and Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101:193–220.

[Matturro et al., 2017] Matturro, G., Barrella, K., and Benitez, P. (2017). Difficulties of newcomers joining software projects already in execution. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 993–998. IEEE.

[Matuleviçius et al., 2009] Matuleviçius, R., Kamseu, F., and Habra, N. (2009). Measuring open source documentation availability. In *Proceedings of the international Conference on Quality Engineering in Software Technology.[cited at p. 23, 193, 197]*.

[Mendez et al., 2018] Mendez, C., Padala, H. S., Steine-Hanson, Z., Hilderbrand, C., Horvath, A., Hill, C., Simpson, L., Patil, N., Sarma, A., and Burnett, M. (2018). Open source barriers to entry, revisited: A sociotechnical perspective. In *Proceedings of the 40th International conference on software engineering*, pages 1004–1015.

[Nagle, 2016] Nagle, F. (2016). Learning by contributing: Gaining competitive advantage through contribution to open source software. *Academy of Management Proceedings*, 2016(1):10856.

[OSI, 2022a] OSI (2022a). Licenses  standards | open source licenses.

[OSI, 2022b] OSI (2022b). Open source definition.

[Panichella, 2015] Panichella, S. (2015). Supporting newcomers in software development projects. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 586–589. IEEE.

[Ågren et al., 2022] Ågren, P., Knoph, E., and Berntsson Svensson, R. (2022). Agile software development one year into the covid-19 pandemic. 27(6).

[Rashid et al., 2017] Rashid, M., Clarke, P. M., and O'Connor, R. V. (2017). Exploring knowledge loss in open source software (oss) projects. In *International conference on software process improvement and capability determination*, pages 481–495. Springer.

[Robillard and DeLine, 2011] Robillard, M. P. and DeLine, R. (2011). A field study of api learning obstacles. *Empirical Software Engineering*, 16(6):703–732.

[Runeson and Höst, 2009] Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164.

[Sholler et al., 2019] Sholler, D., Steinmacher, I., Ford, D., Averick, M., Hoye, M., and Wilson, G. (2019). Ten simple rules for helping newcomers become contributors to open projects. *PLoS computational biology*, 15(9):e1007296.

[Showkat, 2018] Showkat, D. (2018). Determining newcomers barrier in software development: An it industry based investigation. In *Companion of the 2018 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 165–168.

[Stallman, 1998] Stallman, R. (1998). Why free software is better than open source.

[Steinmacher et al., 2014a] Steinmacher, I., Chaves, A. P., Conte, T. U., and Gerosa, M. A. (2014a). Preliminary empirical identification of barriers faced by newcomers to open source software projects. In *2014 Brazilian Symposium on Software Engineering*, pages 51–60. IEEE.

[Steinmacher and Gerosa, 2014] Steinmacher, I. and Gerosa, M. A. (2014). How to support newcomers onboarding to open source software projects. In *IFIP International Conference on Open Source Systems*, pages 199–201. Springer.

[Steinmacher et al., 2014b] Steinmacher, I., Graciotto Silva, M. A., and Gerosa, M. A. (2014b). Barriers faced by newcomers to open source projects: A systematic review. volume 427.

[Steinmacher et al., 2015] Steinmacher, I., Graciotto Silva, M. A., Gerosa, M. A., and Redmiles, D. F. (2015). A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, 59:67–85.

[Steinmacher et al., 2013] Steinmacher, I., Wiese, I., Chaves, A. P., and Gerosa, M. A. (2013). Why do newcomers abandon open source software projects? In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 25–32. IEEE.

[Steinmacher et al., 2014c] Steinmacher, I., Wiese, I. S., Conte, T., Gerosa, M. A., and Redmiles, D. (2014c). The hard life of open source software project newcomers. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE 2014, page 72–78, New York, NY, USA. Association for Computing Machinery.

[Steinmacher, 2015] Steinmacher, I. F. (2015). *Supporting newcomers to overcome the barriers to contribute to open source software projects.* PhD thesis, Universidade de São Paulo.

[Uddin and Robillard, 2015] Uddin, G. and Robillard, M. P. (2015). How api documentation fails. *Ieee software*, 32(4):68–75.

[Venigalla and Chimalakonda, 2021] Venigalla, A. S. M. and Chimalakonda, S. (2021). Understanding emotions of developer community towards software documentation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 87–91.

[Wohlin, 2014] Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA. Association for Computing Machinery.

[Wohlin et al., 2012] Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. (2012). *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

[Yin, 2003] Yin, R. (2003). *Case Study Research: Design and Methods*. Applied Social Research Methods. SAGE Publications.