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

AALBORG
UNIVERSITET

Péter Szatmáry
Maciej Karol Schweitzer
CS-IT
Aalborg Universitet
June 8th 2023

Computer Science

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

Title:

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

Project period:

2022.09.01 - 2023.06.09

Authors:

Peter Szatmary
Maciej Karol Schweitzer

Supervisor:

Adam Alami

Edition: 1
Number of pages: 55
Completed: 2023.06.08.

Preface

> Everyone's reputation is made on a daily basis. There are little incremental things—worthwhile efforts, moments you were helpful to others—and after a lifetime, they can add up to something. You can feel as if you lived and it mattered.
>
> *Chesley B. Sullenberger,*
> *Highest Duty: My Search for What Really Matters*

Abstract

Background. We are all using free and open-source software even if we may not always be actively aware of it. They are increasingly used in consumer software and hardware components as well, therefore a healthy ecosystem of contributors is important for the continued prosperity of FOSS projects. Previous work informed us about barriers that contributors to open-source project may face. One of the hurdles newcomers encounter are various problems with the project's documentation. However, what none of them explores is what are the consequences of these issues detected.

Aim. Here we show through a qualitative case study done on the FlyByWire[1] community, how contributors experience both inadequate and abundant quality as well as quantity of documentation.

Findings. We find how community members coped with deficient documentation through asking their fellow community members for help and making friends with them. Our findings also unveiled, that following the FlyByWire team turning around the documentation's state however, certain members were still asking questions otherwise already answered in the documentation. At the same time, these types of questions were started being answered using links to parts of the documentation in lieu of standard responses.

Conclusion. Ultimately, whether a newcomer will persevere or drop out depends on many factors, however the friendliness of the community, and the state of the project's documentation are two factors that might play a pivotal role in that matter.

[1] `https://flybywiresim.com/`

Summary

Introduction. Our lives are increasingly dependent on open-source software even if we may not realise it. Mobile phones, household appliances, and a considerable amount of servers are running open-source software. For an open-source software community to prosper, it needs a way to both attract as well as retain a continuous influx of newcomers.

Previous work have informed us about numerous hurdles that contributors to open-source projects may face. One subset of these contribution barriers are related to various problems with the project's documentation. Software documentation can consist of numerous parts: user and developer documentations, development environment setup guides, READMEs, contribution guidelines, or notes about the codebase's structure.

Aim. In turn, regarding potential issues with it, documentation can also be outdated, fragmented, too little, or too much; neither of these are ideal serving as a source of knowledge that contributors can rely on. No paper to our knowledge explores what are the consequences of these issues detected, and how they affect the motivation of contributors. Therefore, to shed light on them, we pose the following research question:

RQ: *How does software documentation influence the motivation to contribute in FOSS communities?*

Methods. To answer our research question, we performed a qualitative case study on the FlyByWire[2] community. We think the FlyByWire A32NX project has a considerable research appeal, as they as a community transformed from almost no, to exemplary documentation. To investigate participant's general thoughts and feelings of both states of the project, we conducted seven interviews with community members, both with more senior contributors reminiscing about the project's early days, also comparing it to its current state. To supplement interviews, we performed observations on the community's communication and collaboration platforms. We took a look both at cases of individual participants and through a holistic approach at the whole community in its entirety. Afterwards, we carefully examined the data gathered using thematic analysis.

Results. We found that community members have developed coping mechanisms to cope with deficient documentation in the project's infancy. They did so through making friends with their fellow community members and asking others for help, for information that they could not find on their own. Afterwards, significant efforts were made by the FlyByWire team to turn around the state of the documentation, implementing and improving it. Although the efforts have been made, some members still remain reluctant to do their homework and consult the written materials available. Instead, they still ask questions, questions that have already been answered as well as since been documented down as well. At the same time, these types of questions were

[2] `https://flybywiresim.com/`

started getting answered with links to parts of the documentation in lieu of standard responses.

Conclusions. Writing and keeping documentation up-to-date takes tremendous efforts from the development team, but so does repeatedly answering the selfsame questions. Initially not finding the information however did not deter newcomer from further contributions. They had a passion for the project, in addition to having good relationships with other community members. Fueled by these, did they persevere in their endeavours. Lastly, based on our findings, we have identified a set of recommendations, practical implications for newcomer as well as more senior contributors. For example:

- Newcomer contributors should not feel discouraged when initially not finding written help. Instead, they should let passion prevail, and try to find information through other means. Also, they should do their research thoroughly, try to actually find said information, instead of turning to their fellow contributors from the very start.
- Long-standing contributors should be responsible to bring about a friendly environment encouraging asking for and receiving support.

Table of Contents

Preface **iii**

Abstract . . . iii

Summary . . . iv

Chapter 1 Introduction **1**

1.1 Open-source software . . . 1

1.2 Scope of the topic . . . 2

Chapter 2 Motivation **4**

2.1 Motivation . . . 4

2.2 Research question . . . 4

2.3 Documentation . . . 5

Chapter 3 Related work **9**

3.1 Snowballing . . . 9

3.2 Snowball search process . . . 10

3.3 starting set . . . 11

3.4 Iterations . . . 12

3.4.1 First Iteration . . . 13

3.4.2 Second Iteration . . . 13

3.5 Final set . . . 13

3.6 Related work . . . 14

3.6.1 Non existing . . . 15

3.6.2 Outdated . . . 15

3.6.3 Unclear . . . 16

3.6.4 Fragmented or disjointed . . . 17

3.6.5 Further issues . . . 18

Chapter 4 Methods **19**

4.1 Case studies . . . 19

4.2 Case description . . . 19

4.3 Data collection . . . 20

4.3.1 Interview . . . 20

4.3.2 Observation . . . 24

4.4 Data analysis . . . 25

4.4.1 Thematic analysis 25

Chapter 5 Findings **29**

5.1 Introductions 29

5.1.1 Backgrounds 29

5.1.2 The community 29

5.1.3 Becoming contributors 30

5.2 A look at the early days 31

5.2.1 Guidelines and setting up 33

5.2.2 Version control issues 34

5.2.3 Understanding the project's structure 35

5.2.4 Contradictions 35

5.2.5 Passion and persistence: overcoming challenges 35

5.2.6 Exploring the motivation of contributors 36

5.3 Recent days 37

5.3.1 Intermission: evolution of documentation 37

5.3.2 Artist documentation 40

5.3.3 New people, new documentation 41

5.3.4 The road ahead 41

5.3.5 Documentation matters 41

Chapter 6 Threats to validity **44**

6.1 Internal Validity 44

6.2 External validity 45

Chapter 7 Discussion **47**

7.1 Discussion 47

Chapter 8 Conclusion **49**

8.1 Conclusion 49

Bibliography **50**

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 running on devices. A program's code this way can be either closed- or open-source.

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

In case of the latter, there are numerous similar but not identical definitions of it circulating around; the common umbrella term used is Free and open-source software (FOSS) which incorporates two related but far from equivalent concepts and definitions, namely free software and open-source software [GNU, 2022]. At their core, both promote similar values, such as 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, 2022]. The main difference, according to Richard M. Stallman [Stallman, 1998], the founder of the GNU[1] project and the Free Software Foundation (FSF)[2] is that open-source is a development methodology simply about making the source-code publicly available, 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 as well as the report, we will focus on open-source software as a development methodology based on open collaboration. A methodology that allows anyone, from any country of the world to contribute to open-source projects asynchronously, 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

[1] `https://www.gnu.org/`
[2] `https://www.fsf.org/`

software tools on which we are increasingly depending if even inadvertently. Some great examples of well-known open-source software are the Linux[3] family of operating systems (including the Android[4] mobile operating system), the Mozilla Firefox[5] or the Chromium[6] web browser.

Access to open source projects are generally available through many different repositories, often on self- or internet-hosted Git[7] services like GitHub[8] or GitLab[9]. These repositories usually contain contribution guidelines, issue trackers, and other forms of documentation as well. A contributor, including the project author or owner can expect feedback or suggestions from others on development of new features, bug fixing or on writing documentation and know-how guides.

1.2 Scope of the topic

Learning about new technologies on one's own can be difficult. Contributing to an open-source project can provide the opportunity to expand one's knowledge by 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 new knowledge. In the context of open-source communities, the authors of a project give other people an opportunity to view and modify its parts. Even though core contributors manually review and accept these changes, they can not keep track and remember all of them indefinitely. Consequently, documentation related to the project becomes important in the process of knowledge transfer between developers coming and going.

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

As in the case with closed-source software made by enterprises for profit, an open-source software project have to survive as well. It should have enough members to support it, fix bugs, develop new features. In short it needs to have a healthy community of developers maintaining it.

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 as well as with the retention of existing contributors.

[3] https://www.linux.org/
[4] https://www.android.com/
[5] https://www.mozilla.org/en-US/firefox/new/
[6] https://www.chromium.org/chromium-projects/
[7] https://git-scm.com/
[8] https://github.com/
[9] https://about.gitlab.com/

This process becomes even more challenging as a project becomes bigger and older. As senior developers leave a project, they may not always transfer their domain knowledge beforehand. Also, due to globally distributed nature of open-source software, all communication has to be done online compared to the usual in-office situatedness of enterprises entailing face-to-face communication generally associated with better knowledge transfer. Working in a dispersed team or community usually works asynchronously, it is extremely important to maintain common and clear communication, and in case of open-source software (OSS), some form of shared documentation [Ågren et al., 2022]. Knowledge being lost by not being documented and written down somewhere can, and as we will see is a real problem in numerous open-source projects.

Inadequate documentation can also be considered as a form of technical debt [Li et al., 2015]. Technical debt in software refers to the cost associated with taking shortcuts, making "quick and dirty" decisions during the development process making it easier to get the software working in the short term, but on the other hand oftentimes creating problems or additional work for the developers in the long term. This can entail the usage of bad coding practices, using sub-optimal algorithms, not properly adhering to OOP standards or skipping on important supporting tasks like documenting code being written. These decisions can save time and effort in the short term, but they can create problems later on when the codebase needs to be maintained or updated. Deficiencies in software documentation can make the code more difficult to understand and work with, which in turn can increase the time and effort required to make changes or add new features to it [Li et al., 2015].

Motivation 2

2.1 Motivation

Our motivation additionally 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 tools and frameworks to solve a given challenge. After the respective deadlines, having finished and submitted the projects, we always made time to evaluate and discuss on what should have been done better, what could have been improved and what could be avoided in the future. One prominent common point repeatedly brought up was not properly understanding the structure and the codebase of these 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 machine learning techniques spanning over multiple associated repositories. Unfortunately that project took significantly more time than originally anticipated as well. One of the major differences compared to previous ones was the very limited amount of documentation in addition to the code itself being hard to understand. The lack of useful in-code comments made it even harder to comprehend it. The absence of proper documentation resulted in understanding it being an extremely time-consuming process. Ultimately these experiences have led us to the conclusion that a further examination of the state of documentation and a study of knowledge transfer in general in open-source projects would be valuable. One of our goal is to try to shed light on whether these were due to extraordinary circumstances with select projects, or we might be uncovering a more wide-spread and potentially unaddressed issue plaguing open-source communities.

2.2 Research question

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

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

> *RQ: How does software documentation influence the motivation to contribute in FOSS communities?*

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 a person's motivation to contribute can help open-source projects better support and retain new members, which is essential for the continued health and sustainability of the communites [Hannebauer and Gruhn, 2017].

In order to get answers to our research questions, we will first offer an overview of the literature on the state of open-source documentation and the barriers that contributors, especially newcomers encounter. To identify pieces of literature, the snowballing search process will be used, then we will finish up presenting of our findings.

2.3 Documentation

Documentation in OSS can be divided into two main categories: (i) user documentation helping end-users installing and using the software and (ii) developer documentation that is aimed for those interested in contributing to the software [Bianca M. Napoleão, 2020].

It is necessary to clarify the meaning of the word „software documentation". Manuals, instructions and other resources that facilitate the use of the software. Documentation should guide and help people use the program, or understand the codebase. Poor quality or a complete lack of documentation can result in numerous problems ranging from discouraging newcomers from participation, to abandonment of the project.

In relation to other scientific papers, what constitutes "software documentation" does not seem to have a standardised definition. It means that every single research is based on unique understanding what documentation entails. In a research paper by Nicolas Anquetil [Nicolas Anquetil, 2005] respondents were asked to vote on pre-prepared options. The options were addressed to two different groups of software: structured analysis and object-orientation. In case of the 1st group, they received 70 answers, and an additional 54 for the second. In another paper made by Vikas Chomal, the authors compiled further types of software documentation artifacts as well [Vikas Sitaram Chomal, 2014].

The only common property between the aforementioned research papers is the disagreement of what constitutes software documentation. Therefore in our research, we decided to create our own definition, elaborating on what we will mean when using the word "software documentation" during the course of the thesis. The aim for this is to give readers the same understanding that we have while doing this project. Another important distinction that further reduces potential research papers about software documentation is the difference between for-profit and free software. We choose some of the most well-known open-source projects, and compared their documentation to that of the FlyByWire A32NX: Firefox, Angular, the Linux kernel and the VLC media player. To compare them, we have gathered all types of documentation present in each of them followed by selecting the common types between them. If a given type of documentation was found to be present in more than half of them, then we included it in our list below.

User guide

This documentation artifact is not necessarily prepared for maintainers, but mainly to end-users. It describes how to use program, like as a user's manual. It usually contains a step-by-step guide detailing how to get, download and setup the software, as well as teaching end-users how to use it or interact with it.[1]

Developer documentation

The developer documentation is intended for people who would contribute to the program. It contains a detailed description of the algorithms used in it, the arrangement and operation of individual parts. Numerous documentation artifacts mentioned below constitute part of the developer documentation.[2]

Readme

Readme files are very popularly included with software, especially in open-source project. It is explaining a lot of the following artifacts in a concise, or abridged manner:

1. What does the project contain and what is its purpose
2. Information on how to install the software or library
3. Issues and bug reporting options
4. FAQ
5. License

Installation guide

Under installation or setup guide, there can be two different articles. One for end-users, and one for contributors. They contain step-by-step instructions on how install the program, or setup the development environment as well as how to install the development version of it. It can

[1] `https://angular.io/docs`
[2] `https://docs.flybywiresim.com/dev-corner/`

also include description of the requirements that the target system must meet in order for the project to be run.[3]

Contribution guidelines

This document explains how someone can help with developing or maintaining the software. It contains codes of conduct, detailed instructions, dos and don'ts, as well as recommendations and best practice.[4]

Licenses

Information about the licenses of the project's or external libraries used by the project. A license informs the user about the terms and conditions of how a product is allowed to be used or modified in a legal way.[5]

Issue tracker

A platform or a single document to report and mark software problems, bugs or errors. Figure 2.1 showcases an example of such a system hosted on the GitHub platform.[6]

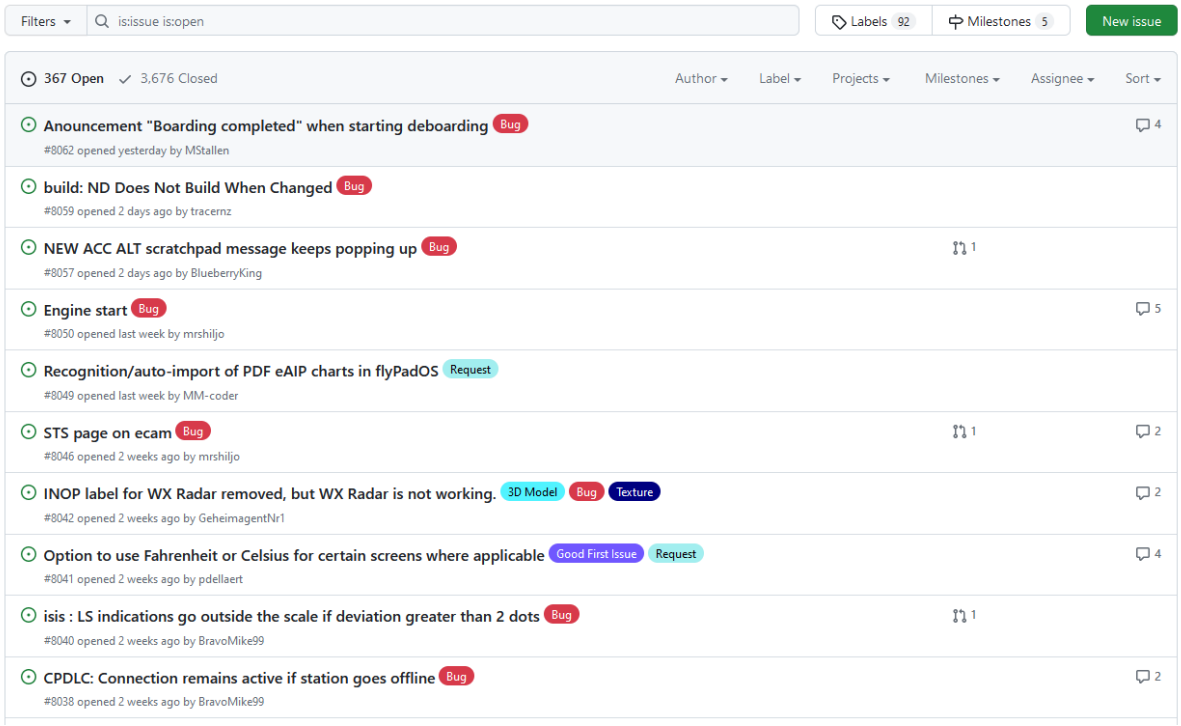

Figure 2.1. GitHub issue tracker

FAQ

It stands for "frequently asked questions". It is a collection of commonly asked questions with answers to them. It is providing help to individuals without having to directly involve other

[3] `https://support.mozilla.org/en-US/kb/how-install-firefox-windows`
[4] `https://angular.io/contribute`
[5] `https://www.mozilla.org/en-US/MPL/`
[6] `https://github.com/flybywiresim/a32nx/issues`

people.[7]

Release notes

Notes containing all information about releases and their dates as well as details of what features were added, changed, fixed or modified starting from the beginning of the software project.[8]

Structure documentation

Index overview of the project folder, as a list of links describing the structure of the codebase, what files, or what parts of the software can be found and where. It organizes information about documents.[9]

Localization process guide

A guide which the describes process of internalization and localization of the project. Supporting multiple languages help the project reach wider audiences. The difference between various localised version can range from just using different languages, changes in the text display mode, for example different text orientation (left-to-right or right-to-left), or even accommodating a different culture approach, like in some cases parts of a content can be offensive for some.[10]

Testing guide / quality assurance processes

This part of the documentation focuses on enforcing standards. The goal is to minimize the occurrence of errors happening and be able to deliver a well-tested and high-quality product. This artifact describes how to conduct tests and looking for elements inconsistent with the project specifics, having unexpected behaviour or crashing the program in its entirety. The testing guide describes how to go about this process, what, when and how to test.[11]

[7] `https://support.mozilla.org/en-US/kb/frequently-asked-questions`
[8] `https://www.videolan.org/vlc/releases/`
[9] `https://github.com/flybywiresim/a32nx/tree/master/docs`
[10] `https://docs.flybywiresim.com/dev-corner/dev-guide/specific/flypad-translations/`
[11] `https://docs.flybywiresim.com/dev-corner/qa-process/`

Related work 3

3.1 Snowballing

To gather relevant literature in support of our study, we settled on using the snowballing search strategy. It is a literature search method in which an initial set of relevant papers is used to identify additional relevant articles, which in turn are once again added to the working set to find further relevant references and citations from[Wohlin, 2014]. This approach can be useful in case there is a lack of existing research on the topic, or the topic being researched is relatively new. By starting with a moderate set of relevant articles and expanding upon it iteratively, the snowballing search method can help us by 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 literature by using what is called a keyword search, we conducted the snowballing search in both forward and backward directions.

During the backward iteration, we identified papers based on references from ones already in our starting or working set, while as part of the forward iteration we gathered relevant articles based on a reverse search looking for articles citing papers already in our working set. We then repeated this cycle until no further papers could be found and added to this collection during the course of an iteration [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 felt that it may be too restrictive in our cases. We thought that using older literature during the first iteration to procure additional potentially relevant papers in the form of forward references may prove to be useful. 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 until reaching a consensus whether to accept or discard it.
3. If such a state could not be reached then we included that article tentatively in our working set until such a time when we could decide whether to include it or not.

3.2 Snowball search process

To initiate 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 collection of literature serving as the 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.

Compiling 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

For the next phase of the snowballing process to identify relevant literature, we developed a search expression using Table 3.1 through a process of iterative refinement. We regularly evaluated the expression to ensure that it still produces 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 roughly 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
("contributor" OR "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 as per the method previously described as "individual then cross-check" that we also used extensively during every phase of the research process.

While analyzing the presented results, we followed a systematic review process, starting with an examination of their titles and then proceeding to the abstracts. Having read them, we then moved on to the introduction, conclusion, and findings sections in that order, and if we were still unable to determine the relevance of the paper at hand, we performed a full text read. As previously discussed, we also applied relaxed time constraints here as well, 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, 2010a]
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 by performing the search iterations. Similarly to the process of assembling the starting set, we performed the iterations individually then cross-checked and discussed each other's work. Each of these iterations 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 newfound ones 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 on 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, using the 18 papers already in 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 approximately 10 years. The references were tightly interwoven, with over 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 to be added.

After having gathered the promising references, we reviewed and discussed them removing unrelated papers. In the end, 8 of the original 18 tentative papers were actually included in the newly created working set for the next iteration.

During the first forward iteration, we collected another 13 tentative papers following 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

Repeating the process described above, we began the second and final iteration using the papers identified during the first iteration in search of references. This round proved to be much less fruitful, with only 3 backward and 3 forward candidates identified. After having reviewed 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 can be considered an exact match as neither of them focuses on the very same topics as ours, 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 in addition to works identifying frequently occurring barriers newcomers and contributors face when trying to join an open-source project.

Previous works generally pointed out four main types of barriers hindering contributors' progress. In this section, we highlight a few of those most commonly occurring ones in open-source projects. In addition, we also going into more detail trying to understand the reasons for their occurences, how they are relevant to our project, and what can be done to counteract the negative effects of them.

3.6.1 Non existing

One of the most glaring issues is if a given project does not have any kind of documentation or documentational artifacts at all; including 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 trying to read the code would be an immense overtaking that not everyone has the time or willingness for, especially if there are alternatives present.

Numerous researchers have, over the last decade identified 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 certain studies point it out [Showkat, 2018].

Furthermore, Coelho and Valente [Coelho and Valente, 2017] found that the absence of documentation pieces can 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 under- or new graduates. It is easy to see why: more senior developers with decades of experience know how to look out for general patterns occurring in software projects and can a lot more easily identify key parts of the code guessing what and how they do then their more junior counterparts. In a later research of his, Steinmacher [Steinmacher, 2015] further confirmed these findings through using various research methods including interviews with contributors and graduate "would-be" contributors.

Matturro et al. [Matturro et al., 2017] also 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 as real issues.

3.6.2 Outdated

It has been demonstrated through other 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, with the program or program code that it is attempting to describe. Aghajani et al. [Aghajani et al., 2019] in their investigation have 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 codebase has undergone significant revisions since its original creation. Oftentimes, documentation is made by a few core or even a single contributor, which then can quickly become outdated as a few key persons stops contributing to the software, or starts focusing on other parts of the project instead

[Aghajani et al., 2019].

Steinmacher [Steinmacher, 2015] found that finding documentation that is no longer relevant can eventually lead to demotivation of users. They found older repositories to be more prone to these kind of hindrances with older projects tending to also be larger doubling down on making the effective lack of proper documentation even more distressing for contributors. They suggest that at the very least, marking documents as outdated should be done to prevent newcomers to the given project from wasting their time with it, 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 even if there are existing documentation artifacts, in case 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 to this family of issues are also cases where the users simply do not have any meaningful ways to determine the creation or latest update time of one, therefore the validity of document fragments that they find. Conversely, overzealous attempts at fixing the aforementioned issues can in turn lead to a situation where there is way too much and overly verbose information available leading to information overload for newcomers, therefore a balance have to be struck between the two ends. In spite of this all, Forward and Lethbridge [Forward and Lethbridge, 2002] argues that some developers can simply learn how to deal with this, 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 very useful for developers to understand the codebase, especially so during their entry into a project [Lee et al., 2017]. In *"A Field study for API learning obstacles"* Robillard et al. [Robillard and DeLine, 2011] mentioned 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 same study of Robillard et al. [Robillard and DeLine, 2011], the researchers also mentioned content-wise incompleteness and ambiguity as other important issues. Both of these can occur 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 it was the case with outdated documentation parts. By definition, an incomplete work lacks a few or

all of the necessary parts for people to make proper sense of it. Incomplete documentation can take on many forms: there are cases where simply certain parts of the software documentation are missing entirely, many other times major parts of a documentation are auto-generated from inline comments, using trivial information not adding anything useful to the discussion that can be otherwise gathered by parsing through the code, e.g. basic signature of a method consisting of it's name, return type, and possible parameters it can take. However, the more complex of a functionality a method has, the more the developers would need clear documentation on do's and don'ts, what side effects and possible limitations it may have and where or how 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 or components therefore making different explanations possible for a piece of functionality. It can leave readers confused about the purpose of a class, method or possible side-effects. The most common issue reported under the scope of this topic was missing examples or 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 or abbreviations not being explained as issues. Furthermore, they also noted that in general, these types of issues are more prevalent among newcomers.

3.6.4 Fragmented or disjointed

These issues are related to the documentation of a software being inadequately presented [Aghajani et al., 2019]. This can surface in the forms of like the documentational artifacts being fragmented between multiple platforms, or although being in the same place having a bad flow making it harder to understand. In their multi-phased study about API learning obstacles, Robillard et al. [Robillard and DeLine, 2011] found that many participants thought that documentation can also severely lack in the department of presentation format and overall quality. They mentioned boilerplate documentation, one-liner explanations that are usually just the rephrased names of methods, as well as overly trivial examples. They concluded that developers vastly prefer a relatively continuous, long and single document to a lot of smaller snippets residing at various places or multiple pages. Furthermore, they also cited that documentation 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-out 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 according to them, newcomers getting lost between them is a real concern.

Robiliard et al.'s [Robillard and DeLine, 2011] participants were similarly 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 function call chain, they summarised. They also expanded upon what is considered a proper, nice flow. The majority of their participants concurred that it can be equally overwhelming to be presented with a large amount of raw, overly formal, precise and long-worded documentation, as it can be disorienting to encounter poorly structured and segmented documentation. What can alleviate these concerns they found, is the documentation having a story-like flow from its beginning to the end expanding upon the concepts in detail as it encounters them while going through the whole program flow. Though they focused on general API learning obstacles, their findings can also be applied in the case of open-source software learning barriers [Robillard and DeLine, 2011].

3.6.5 Further issues

Lastly, there are several other important considerations to be noted regarding documentation in open-source software projects. As an initial point, 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 detailed 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 themselves encounter problems trying to rely on it [Uddin and Robillard, 2015], [Aghajani et al., 2019].

Another documentation-like artifact is the topic of in-code comments mentioned by various authors. They can prove to be a helpful resource as well in helping contributors understand classes and methods. However, 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 also be able to help alleviating fears of newcomers on whether to contribute to the project or not.

These papers all mentioned parts related to our research, as they discussed a fair amount of issues in software documentation, however they failed to mention why these issues are present in the first place together with how do actually these problems affect newcomers' retention in open-source projects. Whether they lead to eventual abandonment, or will new contributors persevere and learn to deal with them through other methods.

Methods 4

4.1 Case studies

For our research methodology, we opted to use case studies. We aim to acquire a more in-depth understanding of the influence that the existence and quality of open-source software documentation can have on contributors to a specific project. Through case studies, we can perform accurate and comprehensive study of our case.

The subject is an emerging open-source software project: FlyByWire Simulations A32NX[1].

4.2 Case description

A32NX

The A32NX Project is an up-and-coming open-source project aiming to develop a high-quality, fully functional, and free A320neo aircraft for Microsoft Flight Simulator 2020[2]. Their aim is to make it as close as possible to the real-life aircraft with the selfsame name produced by Airbus SE[3]. Initially starting out at 2020 august as an enhancement to the default A320neo released with the base 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. Since then, almost all parts of the original codebase was either massively overhauled or rewritten from the grounds-up.

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 4680 stars, 928 forks, and 4074 commits in addition to 2296 approved and merged pull requests

Besides the main repository, since its inception the community has 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.

[1] `https://flybywiresim.com/a32nx/`
[2] `https://www.flightsimulator.com/`
[3] `https://www.airbus.com/en`

The community as of today (2023.06.06) is in very good health. The project has developer as well as thorough end-user documentation detailing how to fly the aircraft in the simulator based on real procedures. 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 for newcomers to find tasks and issues to work on, and development setup guides. For additional questions and quick discussions, the community uses the Discord[4] instant messaging social platform. The average response times can be considered quick; developers who have a question or are stuck with a problem can usually 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.

4.3 Data collection

Qualitative data collection entails obtaining non-numerical data. It is about understanding the feelings, beliefs, values as well as behaviours of people. We have considered qualitative data collection methods including interviews, observations, focus groups, and surveys. [Sutton and Austin, 2015] In order to be able to adhere our time constraints and the globally located contributors of our case, but still 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 performed by asking respondents questions and having them provide immediate responses. Interviews can be conducted in-person, over the phone, or online over video and voice chat. Participants of open source projects projects like the FlyByWire A32NX are not limited to certain countries or regions as it 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 recruited participants on the FlyByWire Discord server that serves as the main communication platform for the project. The message posted on the developer team's channel to recruit participants can be seen on Figure 4.1.

[4] `https://discord.com/`

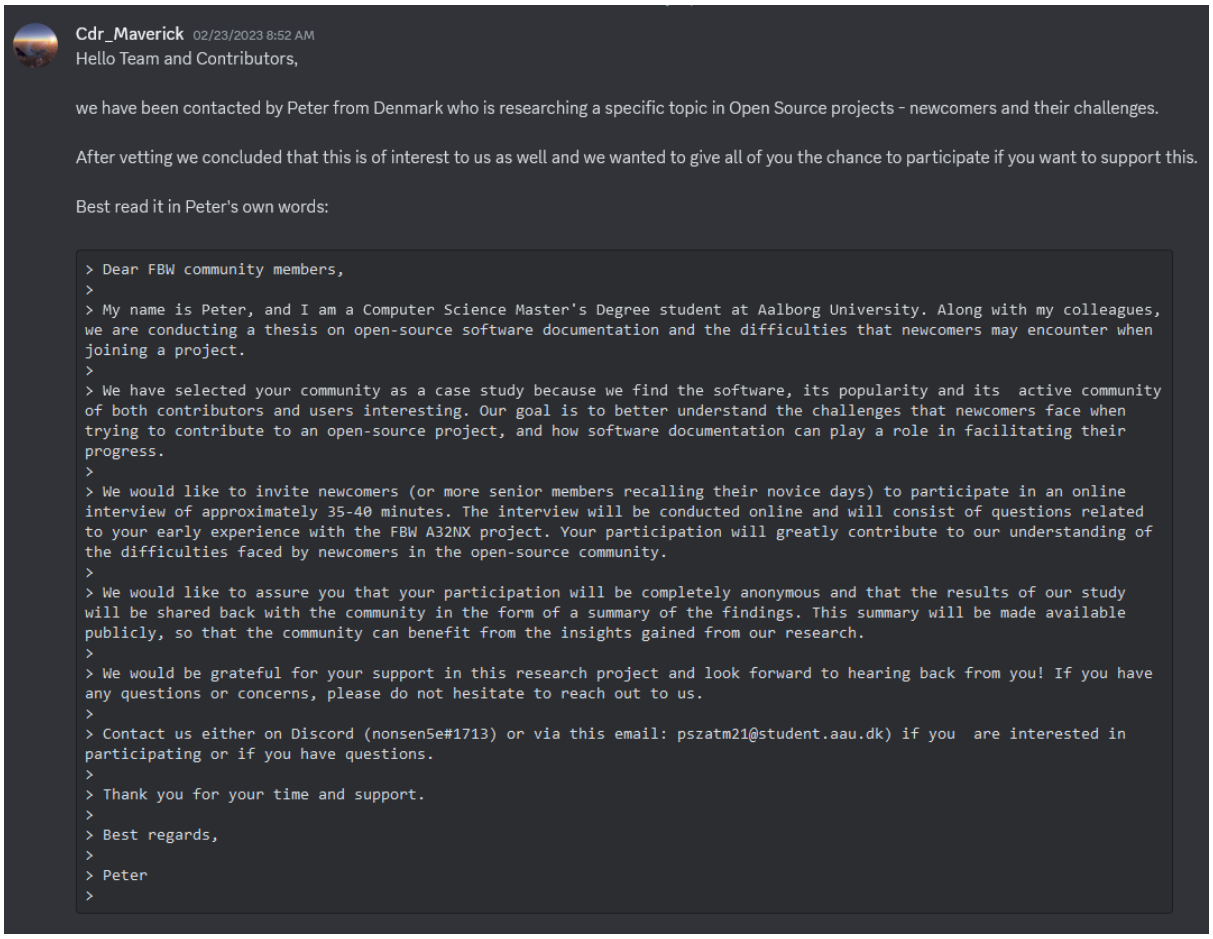

Figure 4.1. Recruitment message posted on the FBW discord

We conducted 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 were open-ended, allowing the respondent to answer in their own words, rather than providing them with a list of predetermined responses to choose from. Also, while we had a list of questions to be asked, they were not strictly asked in order, and the respondent was also allowed to provide additional information or to elaborate on their answers [Runeson and Höst, 2009]. To that end, the set of questions we have composed were serving as loose guidelines, rather than something we had to strictly and methodically adhere to.

The set of questions asked started with introductory ones so as to obtain basic information about the participating community members. The next section of questions were about getting insight into the interviewees early days into the project. The third section is the most extensive as it deals with questions related to contribution and documentation problems. Lastly, the questions at the end are to ask about general thoughts and feelings, also asking participants about other experiences that they may have had with open-source software projects.

The preliminary list of questions we have gathered to that end can be seen below, in the interview guide.

Interview guide

Introductory questions

The introductory questions are about asking about their education and potential previous professional experiences.

- 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 and for how long?

Newcomerness

The second set of question is about recalling their experiences of joining the project, the steps they took to get started, the onboarding process, and their initial interactions with the community and the codebase.

- What motivated you to start contributing to this specific open-source project?
- How did you learn about the project and what steps did you take to get started?
- How did you start your interactions with the community? (mailing list, IRC chats, forums, PRs)? *(if applicable)*
- Can you describe your initial impressions of the project when you first started or tried to start contributing?
- 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 barriers and documentation problems

This part is about problems, mainly problems related to the project's documentation that the participants might have encountered while trying to contribute.

- 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?
- Which part of the documentation do you think is the most important?
- Can you show us an example of documentation and tell how it helped you to contribute?
- Was there any part of the documentation that you were looking for, wanted to see, but could not find?
- 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

These are general concluding questions summarizing and concluding their experiences. Also acting as a double-check re-confirming their answers.

- What other difficulties did you face while trying to contribute to this open-source project?
- 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?

Comparative questions

Finally some optional questions comparing their experiences with FlyByWire to other open-source project they are contributing to or may have contributed to before.

- Was your experience at this project different to other communities? (like succeeding here, but not in others)
- Which community or project was it, and why did you not succeed, or continue working on that?

The interviews were conducted between 2023.02.25 and 2023.04.10. The participants were from all across the globe, so the interviews were conducted on two platforms having online voice chat functionality: Google Meet and Discord. All the interviews took between 30 and 80 minutes. Two online transcription tools were used to transcribe the interviews: otter.ai and tl;dv. Following the automatic transcription process, the interviews were manually checked and corrected when

it was deemed necessary for clarity's sake. Lastly, all the interviews, as well as their participants were anonymised, henceforth we will be referring to the participants by their code names listed in Table 4.1.

Code	Project joining time	Experience	Age bracket
P1	2020	Computer Science Student	<25
P2	2020	SW professional	50+
P3	2021	Unrelated education	50+
P4	2020	Self-learned	25-50
P5	2022	SW professional	25-50
P6	2022	SW graduate	<25
P7	2023	Unrelated education	25-50

Table 4.1. Interview participants

4.3.2 Observation

The technique of observation involves monitoring a person or a group of people as they are going about their activities, trying to or contributing to the FlyByWire A32NX project in our case. Observation is an indirect data collection method - we gather data without interacting with the subjects or manipulating the environment [Lethbridge et al., 2005]. It gives the opportunity to observe behaviors or events that may be difficult to capture through verbal communication.

Precisely because interaction is eliminated from the process, observations can extend data gathered through interviews providing a more objective and less biased 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 about personal problems or failures [Lethbridge et al., 2005] [Espedal et al., 2022].

In this case, we will be conducting direct online observations of a specific open-source contributor or would-be contributor, including their interactions with the community, preferably across different platforms that the FlyByWire community use. The focus of our observations will be the online activities of a contributor, trying to follow and trace them as well as utilize field notes to document their participation.

Field notes of our findings would be taken during the observations. They will help us accurately recording 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 forcing 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 we looked out for:

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

During the research process, we also performed a second type of observation. More overarching, holistic ones compared to the direct observations following a specific contributor's activity. Through them we were not tracing back the activities of select community members, rather we were gathering data on the community as a whole, trying to find trends in their GitHub and Discord chat activities.

The observations were performed between 2023.02.01 and 2023.04.10 with the observed period of activities starting from the project's inception to 2023.03.03 when offline full-text exports were made from the online conversation histories and git logs.

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 focuses on the meaning of themes in parts of the data set, it allows us to fill in the holes; to relate the stories of individual 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 relevant 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, and we are not trying to fit the codes gathered into already existing themes.

We combined the six-phased thematic analysis approach developed by Braun and Clarke [Braun and Clarke, 2006] and extended with using specific coding styles from the two cycle coding methods of Miles et al. [Miles et al., 2013]. The process begins with immersion in the data set, followed by labeling or coding of it, and then grouping codes together and shifting to themes that captures and represents some key concepts relevant to the research question. Finally, a report of the findings can be made using the themes identified this way.

The approach used is as follows:

Phase 1: Familiarization

The first phase is to get immersed in the data becoming thoroughly familiar with it. This can be achieved by going through the raw data multiple times, listening to audio or watching video recordings, reading and re-reading the recording transcripts and the field notes repeatedly. However, merely reading is inadequate by itself. One has to think about the meaning behind the words. Now is a good time to also make jottings about relevant aspects related to the research question. [Miles et al., 2013]

Phase 2: Generating initial codes

After getting intimately familiar with the data set, the next phase deals with labelling or coding it. In Miles et al.'s [Miles et al., 2013] approach this is the first cycle of coding. 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, usually the most important part from a data chunk. 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. Keeping a code short is of paramount importance here. Longer descriptions, or full-blown summaries of data segments can make it harder to group them together in the following phases.

During the first cycle, there are many different ways to code a piece of data. Miles et al. [Miles et al., 2013] defined twenty-five different first-cycle coding methods or styles, based on which one can code the data. We opted to use four of them:

Descriptive coding
Probably the simplest and easiest form of coding. Descriptive coding summarizes chunks of data into short phrase, rarely more than one or two words long. Codes created this way are usually simple nouns. *"newfound passion"* and *"convoluted documentation structure"* are examples of descriptive codes from our transcripts

Value coding
This methods focuses on the values: feelings or beliefes of the participants. This type of coding allows us to take insight into the participant's "perspective or worldview" as defined by Miles et al. [Miles et al., 2013]. Examples of value codes are *"enjoying development"* and *"loves supporting others"*.

In Vivo coding
In Vivo coding is based on the use of short words or combined phrases in the data, directly quoting what the participants said. *"started as a mess"* or *"became friends"*

Process coding
This method use gerunds to showcase changes, actions or activities happening in the data. Like *"asking a lot"* and *"struggling"*

Phase 3: Searching for themes

With the pre-processing steps out of the way and having performed the first cycle coding on all the interview transcriptions, we make a shift from raw data to coded data, and from coded data and codes to themes finally. The codes already included an initial level of interpretation in the previous phase, but more as a side product of creating summaries for labels, of summarising blocks of data to a few words. This is the phase where we really focus on the interpretation of the data. The upcoming three sections encompass Miles et al.'s [Miles et al., 2013] second cycle coding. 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 it identifies and explains the main topics of interest; grouping together and summarizing them. In this phase we can also start considering the connections and relationships between themes as well.

Introductory			
prior experience	**motivation**	**community positives**	**new beginnings**
first programming community	chaotic order	chaotic order	annotating good first issues
inexperienced contributors	coordinating work	coordinating work	asking a lot
university student	enjoying development more than playin	enjoying development more than playing	newcomer difficulties
still a long way to go	gratification	gratification	low hanging fruits
self learned dev	having fun	having fun	nothing to do
experienced developer	project philosophy	project philosophy	onboarding

Issues			
help and struggles	**codebase**	**contributingh issues**	**version control**
doing your own task	code standards	keeping up with changes	accepted PRs
do it yourself	coding knowledge	beginner's guide	looking at PRs
developer guidance	complex to understand	3rd party dependencies	first contribution
receiving feedback	convoluted structure	API workaround	pull requests
always helping out	technical debt	aviation knowledge	no git education
searching for answers		setup	PR comments
looking at PRs		texture/model frustration	

Documentation			
documentation	**development guide**	**documentation issues**	**causes**
what makes a good documentation	documenting the structure	docs at multiple places	don't like to write documentation
docs close to the code	hello world example	badly organized	badly organized
improving documentation	setup guide / dev env docs	contradicting docs	no manpower
reworking docs	user documentation	straight, to the point	code changing fast
reading documentation	how to start	docs becoming outdated	hard keeping it up to date
taking the lead		non existing documentation	turnover
			dedicated team

Figure 4.2. Results of cycle 2

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 they are indeed supporting their points. In order to ensure that the themes are of good quality, we need to consider several factors, including whether they are well-defined with clear boundaries and a singular focus, whether they are backed by a sufficient amount of data points, and whether they directly address at least one aspects of the research

question.

Phase 5: Defining and naming themes

It is like a second follow-up quality assurance phase. The goal is to improve upon the themes we settled on using. This is also the place where we interpret the themes' data, draw conclusions from them, and start preparing the overarching narrative of the story; how and when to include the themes and how to connect them together. Figure 4.2 showcases an abridged excerpt of the results of thematic analysis showing similar themes grouped together in addition to similar codes combined under named parent themes.

Phase 6: Producing the report

Lastly, a detailed report is made in the form of a compelling story telling our findings in a clear and straightforward manner. The result of this phase is Section 5 about our overall findings.

Findings 5

5.1 Introductions

In this first sub-chapter of our findings, we first introduce the community as well as its members with special focus on the interview participant's backgrounds: What motivated them to join the project and the community, what steps did they took to get started and what are their overall feelings about the project.

5.1.1 Backgrounds

The FlyByWire community includes over two hundred members with diverse backgrounds; they are either part of the core development team, or just be in the periphery helping out once in a while. Their experience levels varies from non-experienced contributors, hobby developers who are just starting out in the world of programming with no prior coding experience all the way to professionals working in the field of software development for decades. In addition, there are also students taking courses in computer science having some programming knowledge, and there are also self-learned developers with no or unrelated formal education having acquired the necessary knowledge by themselves through online learning resources or by participating in OSS development projects. Some of them are first time OSS contributors, while others are long-time regular contributors in other open-source projects.

The community also has a broad range of contributors of different ages spanning from university student young adults, to middle-aged and more elderly contributors nearing retirement. Lastly, the geographical location of the contributors are also varied having contributors from Europe, America, and Australia even.

5.1.2 The community

Likewise, their motivations also varied ranging from the desire to learn new skills, learning new technologies or getting more professional experience to just *"doing it for fun"* or giving back to the community. What is common in all of these people however is a collective passion for aviation as a hobby, their shared love to get involved and contribute to a project that they all use, which in this case is the FBW A32NX aircraft within the Microsoft Flight Simulator environment, *"kind of a little bit addictive, I would say in a positive sense, obviously."* as P2 stated.

What is also common across the interviewees, how they all mention that overall how friendly the community is to new members, citing the immense amount of help and support they got when starting out, trying to answer any and all questions, providing support and helping out wherever they can, quickly making friends in the end. *"I quickly made friends with a few of these guys, probably this, that is the main reason I am still here and doing what I can"* as P0 said. As a counterpoint, two participants mentioned personal issues and conflicts with others, as well as controversies erupting between members. In addition it was repeatedly pointed out, that non-contributor end-users can be seriously toxic, requesting status updates, new releases and bug fixes in an entitled and demanding manner. It was later verbalized to us as well, that that the stress caused by this behaviour one potential cause of contributors leaving the project.

Oddly enough, one point being brought up by a member that how bureaucratic the whole contributing process is. Another member on the contrary refuted it by saying how much they love the degree of freedom, and equality between members. In their own words: *"I'm actually surprised how much inherent dynamic there is to such a project and how little actual management"* (P2), and *"there's very little leadership or playing, we don't need that we actually work on consensus."* (P2). This was characterised later on by him saying that the community is operating on *"chaotic order"*.

5.1.3 Becoming contributors

As previously discussed, the original incentive beside the inner passion for joining in the development of the project oftentimes stemmed from a personal desire for involvement, like through finding about a bug or incorrect behaviour and setting out to fix it.

The main platform of communication and collaboration of the FlyByWire project aside from GitHub [1]GitHub is Discord [2]. Discord is a social platform where people can communicate on text, voice or video channels on servers, share images or send files to each other, as well as share their screens to groups. Originally it was created especially for online gaming communities, however it quickly got adopted by other online communities including open-source projects, and even by classrooms during the COVID-19 pandemic due to its rich feature set and ease of use. [Mock, 2019] [Vladoiu and Constantinescu, 2020] [Kruglyk et al., 2020]

This Discord server was present right from the very beginning, serving as the go-to platform for users and contributors alike, naturally also serving as a starting point for all newcomers to the project, be them just end-users looking for a community of like-minded individuals, contributors, or would-be contributors looking for help with starting contributing or having an issue with development.

All of our interview participants started their contributing careers in Discord one way or another.

[1] `https://github.com/`
[2] `https://discord.com/`

As discussed above, certain members joined the community to correct some issue with the software that they encountered:

P3 encountered some unexpected behaviour with the software, then went on the Discord server, and asked whether they can fix it. Also saying that he thinks it would be a very easy fix. The respond he got was *"Why don't you do it yourself?"*, and so that is what he did, setting up to fix the issue himself. Later on both this, and multiple other pull requests of his were accepted.

P4, who worked as a model and texture artist for the FlyByWire project joined under similar circumstances, not being satisfied with how parts of the aircraft model behaved, joined the Discord, and noticed that the project was actually looking for knowledgeable modellers as well.

Other team members in the meanwhile have started their initial involvement by offering help to community members regarding the use of the software (i.e. how to use the various functions of the airplane in the simulator).

P7 joined the team at a much later stage, by what time most of the early issues of the project has been already fixed. His process was not that different either, originally started interacting with the community, asking and engaging in discussions as a user, then started out helping out his fellow users more and more himself, followed up by trying to fix bugs in the codebase.

There was an unanimity between them agreeing that without it or a comparable platform, the community would be nowhere now, as P2 stated *"I'm totally, totally convinced that without discord or something similar, this project would never have happened."*.

We can see that in the vast majority of the cases, the progression from peripheral involvement to being an integral part of the core contributing team was typically a fluid transition, a *"fluid thing"* as we have saw a Discord community put it during our observations. It was not based on recruitment, by previous achievement or passing some arbitrary test. To become a core member of the team, the things a newcomer needs is passion and persistence in making contributions to the project. However as we discussed, this road to becoming a contributor was never without hurdles.

5.2 A look at the early days

Here, in the second sub-chapter we take a look at how the FlyByWire A32NX project looked like in its infancy. We explore what was the general state of the codebase and the documentation in addition to taking a look at what types of issues participants encountered trying to contribute to the project and how did they try to overcome them.

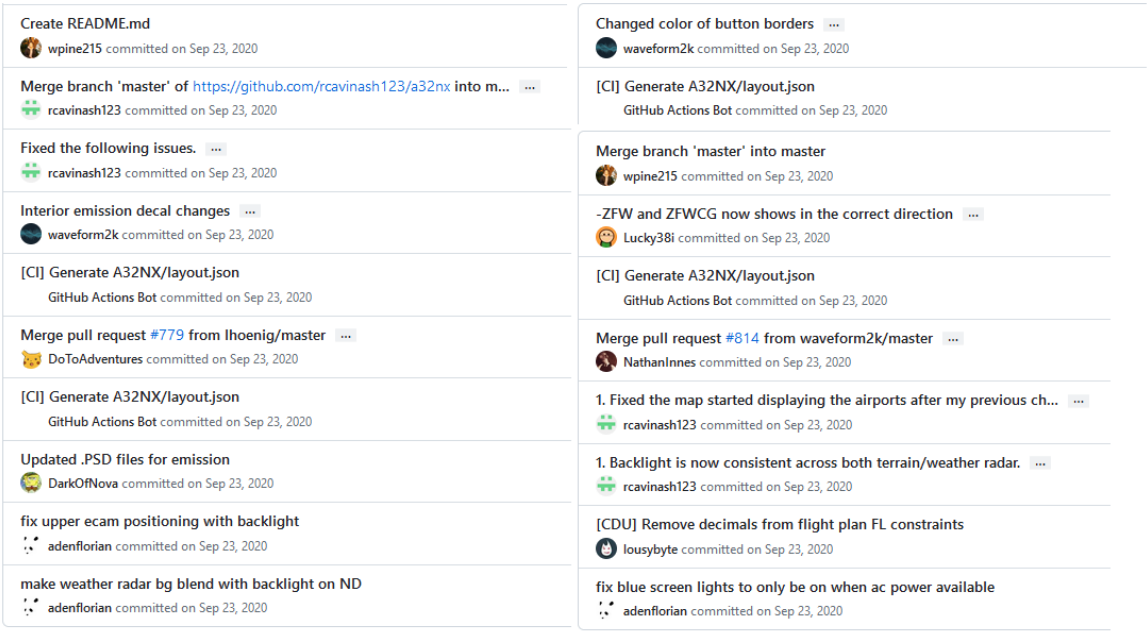

Figure 5.1. A days worth of commits during the early days of the project

In the very beginnings of the project, development quickly picked up with breakneck speed as lots of people were joining the project. This can be seen on Figure 5.1 showing a single day's worth of commits during the second month of the project. Of course, there were a lot more tasks easily being done by newcomers, more low-hanging fruits, and a great number of people did lots of works on various parts of the system. This is not to stay, that the project did not have issues.

In communities, people naturally have conflicts; conflicts about which way to implement certain things, what to implement or not implement, and so on. People also voiced their frustration regarding their lack of knowledge of how to contribute, what or how to do in the project. Younger newcomers also expressed their that starting out is or was much more difficult than they had initially anticipated. *"I didn't expect it to be that difficult"* (P1) and *"To tell you the truth I thought it would be easier"* (P6). Challenges the newcomers faced were ranging from having no clue where and how to start contributing and what to do to having lack of platform or project-specific expertise, or insufficient general programming-related knowledge in the select languages, platforms and systems used.

Their feelings regarding the project's (the codebase's) state were described as *"frustrating"* (P7), *"hard"* (P6), *"pretty difficult"* (P1). Quite a few contributors expressed their frustration with the codebase saying it is complex, hard to understand, navigate, it's structure is convoluted, and could use a restructuring. At this time, the community was still figuring out how the simulator platform itself, or the originally provided software development kit (SDK) works. The documentation landscape was in comparable shape, knowledge regarding the project and

platform was not yet widespread or written down with P3 saying it was *"extremely minimal"*.

Not knowing the project's philosophy was another issue that was brought up during our research. Newcomers started passionately developing and adding made-up features to the A32NX aircraft that were not in line with the project's philosophy of *"if you create a feature, you need a real life reference"* (P2) meaning if it's not in the real A320neo aircraft done by Airbus SE, it will not be in the FBW A32NX aircraft.

There were several obstacles newcomers had to overcome. We gathered a few of the most prominent ones below, a few of which we will also explain in greater detail:

- Problems with how to start
- Troubles setting up the project locally
- Problems understanding the base platform
- Not knowing how to use the VCS
- Contradicting parts

At that stage, the main source of obtaining information was getting it through personal, word of mouth communication from their fellow developers. People asking questions were sometimes suggested lacking other options to find out how the general development workflow is done by looking through commits, PRs and issues. One contributor asked *"Any guides on how to get up to speed with developing systems etc? I can't seem to find good information on anything"* and the answer they received was *"I would suggest looking through the commits, PRs and issues to get familiar with how development is done."* On another occasion, someone answered another's question in a similar manner *"I basically learned by looking at existing XML. And copying and modifying it. You can look at my old PR's to see how I did it"*. Likewise, P3's experience was alike as well when asked about how he familiarized himself with the codebase and the general way of doing things. He said *"So I looked at other people's PRs and what they did to achieve this and that, describe what they've done, you can see the old code and the new code and compare it to something. That was how I learned really"*.

5.2.1 Guidelines and setting up

Supported by both issues reported by our interview participants as well as by our general observations, the most glaring issues were lack of proper guidelines for project initiation and setup as well as about the general development workflow. This topic encompasses a whole host or family of issues which were raised both on the various development related channels of the Discord as well as the interviewees:

- Setting up the development environment itself, including runtime platforms and IDEs, like Visual Studio, Docker and Virtualization in the operating system. *"Docker was the biggest issue I think"*(P3).

- Downloading, configuring and building the software locally. *"Is there any guide for new developers on how to get started and get a compiled version of the aircraft going?"* Discord users asked multiple times.
- Project specific git workflow questions, like what branches to work on, when to submit a PR, a draft PR, how to format it, and where to list features or bugs one has started to work on. *"the percentage of people who don't actually know how to use Git and GitHub is higher"* (P2)
- How to use and list references for features being implemented. "Guys, I need a reference document for the PDF and ND for the A320" (Discord member)
- How and when to do tests. P4 retrospectively praised the importance of documenting the testing process the following way: *"In the beginning, we didn't even have that many QA testers. And at some point, when we did have QA testers, we did include how they should do QA tests. And I think that's actually one of the best things that fly by wire did".*

5.2.2 Version control issues

Another frequently observed and reported issues however were one way or another related to either Git and GitHub. Specifically pertaining to general Git knowledge that could be attributed to inadequate Git education. In contrast, project-specific Git workflow issues were relatively infrequent, but still present as we discussed above. This phenomenon is a lot more prevalent among those without software-related backgrounds, as well as computer science students and fresh graduates. As P2 mentioned, *"I knew how to work with GitHub. This is, for example, a very important skill, a lot of newcomers have issues with like, what's a PR how to do a PR?".* P3 having no programming experience before also recalled as such: *"I must have tried to clone it or fork it maybe five or six times until I were successful. Just for some reason, it would just be some some mismatch and it wouldn't run and I'd try again".* Finally, newcomers from the dev-support channel of the FlyByWire Discord echoed similar sentiments: *"how do draft PRs work? I want to open a draft PR"*, *"lol github is hard to understand - I've tried a few times, watched like 30 min youtube videos on it, and STILL don't really understand"*, and *"How do I update my fork for making a PR? I have 'This branch is 574 commits behind flybywiresim:master'"*, lastly a contributor experiencing some degree of success in the end *"alright, a bit of rebase hell, merge master chaos, manual copy pasta from old changes ... the one that should work :)".*

Even though these types of issues are outside the scope of both this work and the FBW project documentation; as of 2023 spring the documentation maintainers of the project are actively contemplating adding a Git and Git workflow guide to the project documentation as part of the contribution guidelines to help with the seemingly endless flow of git related questions on the support channels.

5.2.3 Understanding the project's structure

As development continues on at great speeds, it can prove to be very hard to keep up with changes on the documentation side of things. It also goes without saying that as the size of the project grows, it gets harder to keep track of the structure; both of the codebase and the documentation, or even keeping it in check enforcing best practices. This in turn makes it increasingly harder for newcomers to join a select project. This claim was further supported by P1 saying *"the structure the entire thing is convoluted. Even the main, so that it's very hard to navigate. But I guess that was the one main piece of documentation I was looking for"*, and also by observations of ours where community members were asking for locations of certain files or functionalities such as *"Does anyone happen to know where the code lives that specifies what livery is used for the default/thumbnail livery in the aircraft selector"*, and as *"do you have any manuals or instructions for the code? Or do you sit up at night looking for variable names? did not understand what is responsible for what and where"*.

5.2.4 Contradictions

Contradictions are another types of issues that can be attributed to the speed the project was developed with. At great speeds, it gets almost impossible keeping up with the changes, and information regarding select parts can be outdated even before being fully developed. If there is no communication between developers and/or between documentation maintainers, it might result in outdated or contradicting parts present, or the same thing being written down at multiple places making people even more confused than without. In early 2021, several different documentations sites were set up as well as linked on the newly created documentation channel until the community settled on the one still being used. Illustratively a contributor also said *"to get back to the docs, i feel like simbridge docs are kinda hard to navigate. i don't know how to fix it right now but i feel like one issue is the fact that certain pages are simply very long. most pages are pretty concise but 'Remote MCDU Display' and 'Troubleshooting' are just super long."* and another one got confused by outdated documentation parts not marked as such expressing their findings as such: *"Seems I was using an outdated document (https://docs.flybywiresim.com/fbw-a32nx/a32nx-api/a32nx-flightdeck-api/) but found this one now searching for that variable: https://github.com/flybywiresim/a32nx/blob/master/docs/a320-simvars.md, Guess the latter one is the most current one?"*.

5.2.5 Passion and persistence: overcoming challenges

Even developers do not necessarily care for writing documentation, *"developers don't like to write documentation"* as P2 phrased, or even reading it. On top of that, manpower was also lacking - members wanted to contribute to the actual code base, implementing new features, or fixing bugs that they found - instead of writing down how the code works. Others has stated, it is due to a rate of turnover, and lack of dedicated team responsible for maintaining it. In our case, all

of our participants agreed that reading documentation can be very useful, whether it being the overall faster way of obtaining information, or just simply being the more independent way of doing things, not being dependent on others. Our respondents agreed, that they would all try to get the necessary information on their own first, followed by asking fellow community members if needs be. *"I hate being one of those people who are asked a question that's already been asked. So what I would do is either touch up with Discord's search function, like my question or like, something along the lines of my question and see if it's already been answered. And if it has, then that's great. I'll just use that answer. And if that answer didn't work for me, then I'll ask it again."* stated by P1.

However as we perceived it, during the early days of the project, the documentation was not yet very well developed, leading to a lot more asking more senior contributors for information that what would have been necessary otherwise.

In conclusion and in spite of all of the aforementioned problems with the documentation, we could not see a direct trend between newcomers leaving or dropping off contributing entirely and documentational issues.

5.2.6 Exploring the motivation of contributors

What can be the potential underlying factors explaining this?

Unsurprisingly, more experienced developers with decades of prior experiences reported documentation issues as less of a priority, citing going without documentation comparatively less of an issue. Their accumulated experience made them understand the codebase much more quickly on their own. They too are asking questions whenever something pops up, but are far less prevalent on others help to being able to navigate and comprehend the project structure with ease.

Younger and inexperienced newcomers have however developed coping mechanisms. [Kelly, 2004] Coping mechanisms that helps them through the undeniably hard days that is the state of being new to a given software project. Their solution was all about overcoming challenges by finding help from others. As we previously discussed, making friends with community members was a phrase mentioned by all of our interview participants, *"Interacting with some of the developers there became good friends with a lot of the team"* (P1), *"positive spiral"* (P2), *"positive attitude"* (P5) and *"good community"* (P6) and similar thoughts were brought up repeatedly. P6 mentioned *"i'd much rather find the infos on my own, but hey if i couldn't, what else can i do?"* in regards to repeatedly asking questions *"pestering"* the development team. To get information that they could not find out on their own, they had to ask questions, a lot of questions from their fellow community members, from their fellow friends. And questions they asked, both in private chat channels as well as we learned it from the interviews and in public channels encouraged by the FBW team members, to make the questions and answers also as a source of information

for future members potentially asking the same question later down the road. And more often than not, answers they received in turn. This is greatly supported by our overall observations as well, that less than approximately 10 percent of the questions asked on the development support chats goes unanswered. Response time also varies greatly, from minutes ranging up to a day - obviously due to the global reach of the project, and time zone differences. P7 very nicely summed this phenomena up as *"using others as human documentation"*.

At its core, even though most participants would liked to have more written documentation to help them start with the project and struggled, their passion for the project and community was bigger than the demotivation caused by not knowing how to start, and through their passion and repeatedly using the collective knowledge of the community, they managed to onboard themselves, start contributing and joining the development team later on.

5.3 Recent days

Last, we are turning our focus on the project's near past, present and future. How the development and documentation team turned around and improved the documentation's state, how do contributors who have only joined the team recently experience its current state, finishing up with discussing what the future has in store.

5.3.1 Intermission: evolution of documentation

The main documentation repository of the project what we can see currently here [3] from which the documentation site[4] is automatically generated was created at early 2021. This is not to say, that the project was without any kind of documentation beforehand. Rather, it was minimal, present at multiple sites, and a great deal of it was .docx files shared around and pinned in Discord channels. Even until recently a big chunk of the documentation lived in a folder inside the main A32NX repository [5], which have been now replaced with a single markdown file detailing the repository structure - a feature that was asked recurrently, and even was mentioned above.

[3] `https://github.com/flybywiresim/docs`
[4] `https://docs.flybywiresim.com/`
[5] `https://github.com/flybywiresim/a32nx/tree/82f9ad7e2802172d706fa08ac051597f1a9b8e11/docs`

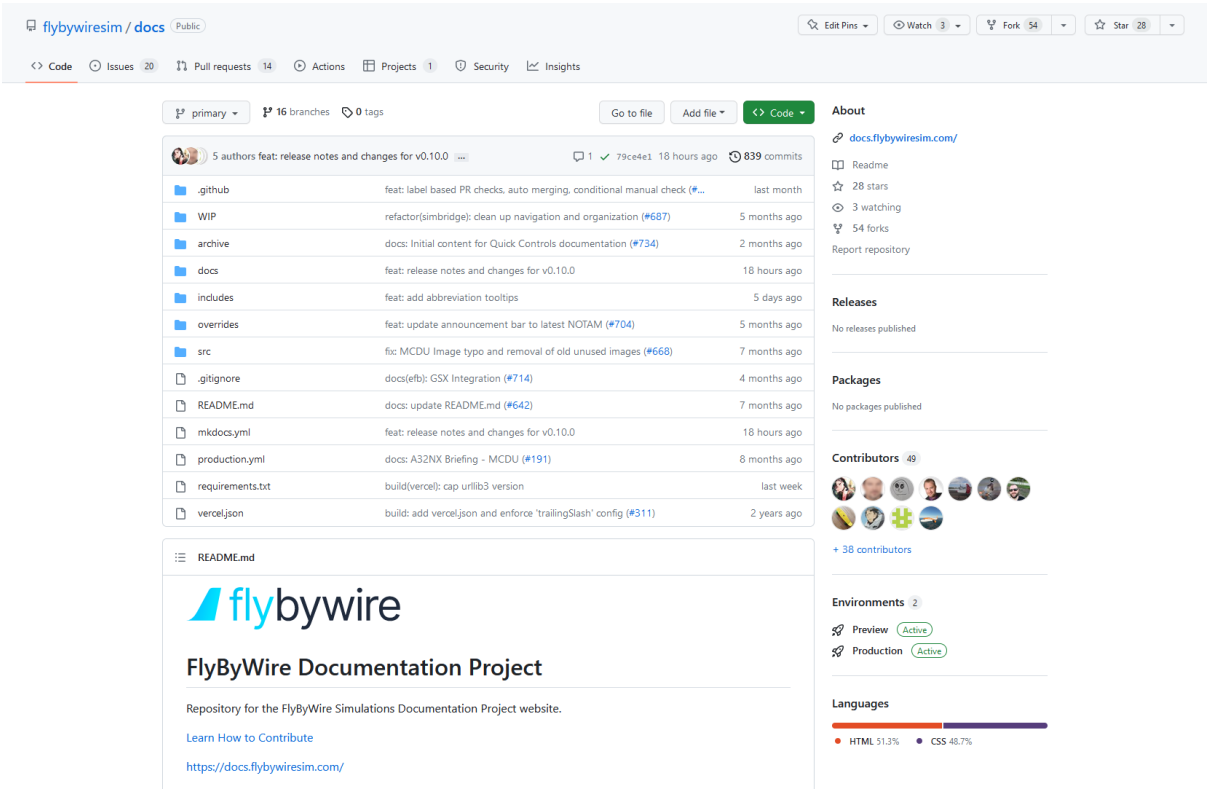

Figure 5.2. Documentation website

Coming in after the initial few months of development on the project, some community members were spearheading the documentation develepment effort. In certain cases, members took the lead themselves. In other cases, members started on their own volition, simply noting down their own experiences setting up, or documenting parts of the API to help themselves, and potentially others with the development process later down the road. The core development team having noticed this approached them and asked whether they would be interested in helping out the documentation efforts. Thus a relatively small but dedicated team quickly grew around this initiative in creating a dedicated documentation repository that should serve as an integrated knowledge center for both users and developers replacing the disorganized and decentralized documentation fragments that were present before. In the subsequent link, one can see the "docs" subfolder of the main repository on 31.12.2020 representing its early state before measures has been taken to create a separate documentation repository). [6]

[6] `https://github.com/flybywiresim/a32nx/tree/ad7146122e7852052e39a4a6bd510a9f5c101043/docs`

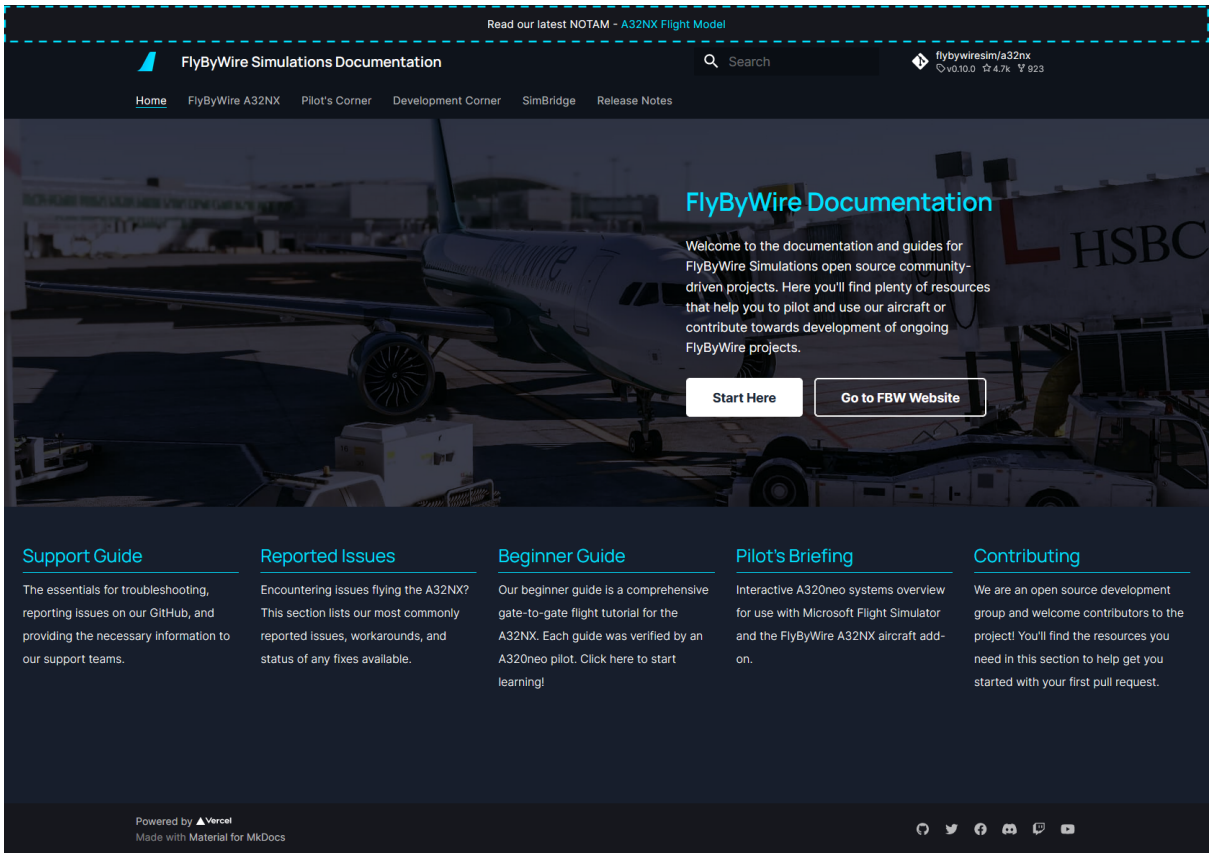

Figure 5.3. Documentation repository

The team created more detailed guidelines for contributing to the project, including tips for developers, specific subsystem development guides, and written down the established testing and quality assurance protocols. In addition, they created a separate web page that dynamically tracks changes in the documentation, displaying them in a visually appealing manner. The new "docs" repository can be seen on Figure 5.3 and the documentation website being automatically built from it is presented on Figure 5.2.

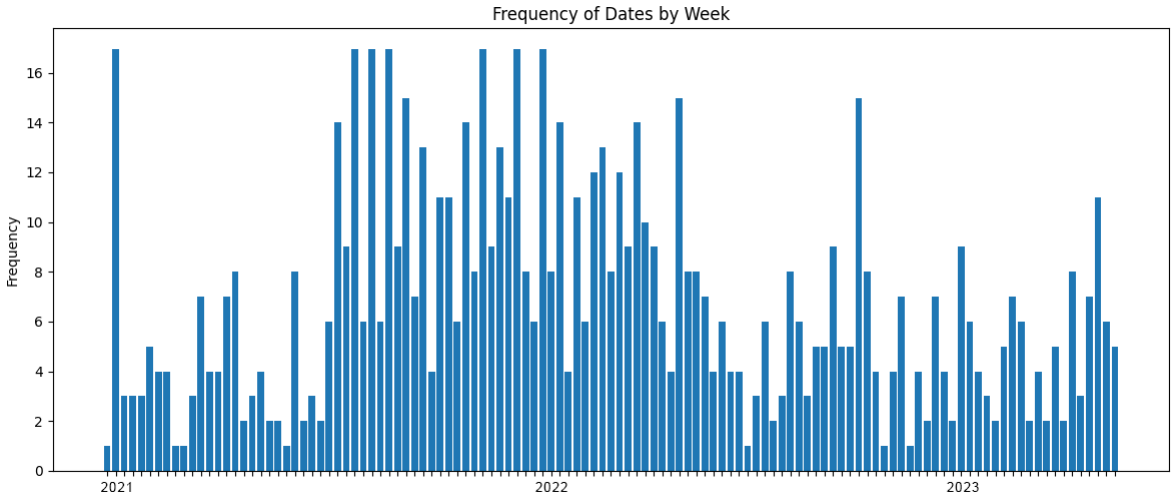

Figure 5.4. Number of documentation commits by week

Figure 5.4 showcases the number of documentation repository commits done by week, equalizing the outliers by capping them at 3 times the standard deviation value of the data. The outlier days diminished that way were individually inspected by the authors, and were found to be always just a quick burst of testing, or typo or spacing fixing commits done by the same developers. We can see that after the initial start, the commit frequency slowly ramped up and stayed consistently very high for almost a year having reached its nigh-current form during that time frame, but activities also stayed relatively consistent ever since.

5.3.2 Artist documentation

During our research, it also came to our attention, that the FlyByWire team comprises not only of software developers, but also texture and model artists contributing to the development of the product. Model artists are creating the 3D objects, the frames of the airplane like its shell, the engines, and the various screens and buttons in the cockpit. Textures artists create "skins" that are applied on the aforementioned objects surfaces making them detailed and feel realistic. Unlike software developers however, they do not necessarily require documentation as their work is standardized by the tools they use in their respective industries. P3 answered when asked about specific knowledge required or potential documentation that would be welcome in this area that *"You don't really need because you can, you can technically just like modelling in every game is always the same. It doesn't really change"*, as well as *"take the same one that you just did for FlyByWire and import it somewhere else. So yeah, it's, it's the model itself does not change"*.

In their case the FBW project currently contains only supplementary information in the form of useful tidbits for these artists, available as part of the development guide section of the documentation.

5.3.3 New people, new documentation

Overall, people have expressed their satisfaction towards the changes continuously being done during the years. Long-time contributors have agreed, that the changes and the ongoing development is definitely welcome, however this is not to say, that overall newcomers have an easier time to contribute nowadays. On one hand, due to the accumulated knowledge over the last three years or so undoubtedly makes it easier for them to start up, help them knowing how or where to add new features, or make sense of the project files on the whole. On the other hand, these measures only mitigate the additional complexity the codebase have accrued during that time. Now more of the complexity lies in actually planning and implementing one's own contributions, understanding line by line code of subsystems, rather than in getting started. Newcomers P5 and P6 who have joined the development more recently has especially expressed their positive feeling when asked by saying *"i think it's alright. i mean i found everything i wanted i was looking for"* (P5) and *"better than elsewhere"* (P6).

Good documentation makes contributors' lives easier. First, it helps newer contributors to the project by them not having to discover all the elaborate inner workings of the codebase by themselves, and also by not having to wait for substantial periods of time for answers for their questions asked on a discussion board. More knowledgeable members may be busy for extending periods of time, or live in very far timezones from them. Likewise it also helps well-experienced long-time contributors of the project by them not having to spend their precious time answering the same questions allowing them to focus more on planning and development of the project.

5.3.4 The road ahead

In line with our prior discourse, though the documentation has improved a lot during the years, there is still much work to be done. The issue tracker is ever not empty and the documentation team is ever thinking about creating new, improved or more in-depth guide for the users as well, or just trying to make the whole structure ever more logical. *"This is completely recent, when I did the repo restructure. I said I started with that project in December, restructuring the repo. And when I did that, I also did this documentation."* (P2). The fact had not changed either, that many developers simply do not like writing documentation, the codebase is continuously growing as well, so documentation maintainers have to always keep an eye on the project, looking for changes therein.

5.3.5 Documentation matters

From an outsider perspective it seems not much has changed overall, but it could not be further from the truth. Even though the project's users have persisted despite sub-optimal or non-existing documentation, having high-quality documentation is crucial for the long-term success and growth of the project. A great chunk of then commonly asked questions are answered in the FAQ section as well as the documentation containing many previously requested parts. For

example in multiple cases community members on Discord were asking for guidelines on how to start *Where should I start if I would like to explore the development work. what is the next step?*, *Does someone need some kind of dev support? I am searching for a way to start contributing.* and also a lot of questions about from end-users ranging from how to install the software to how certain parts of the aircraft work, many of which are now answered in the common questions[7] section as well.

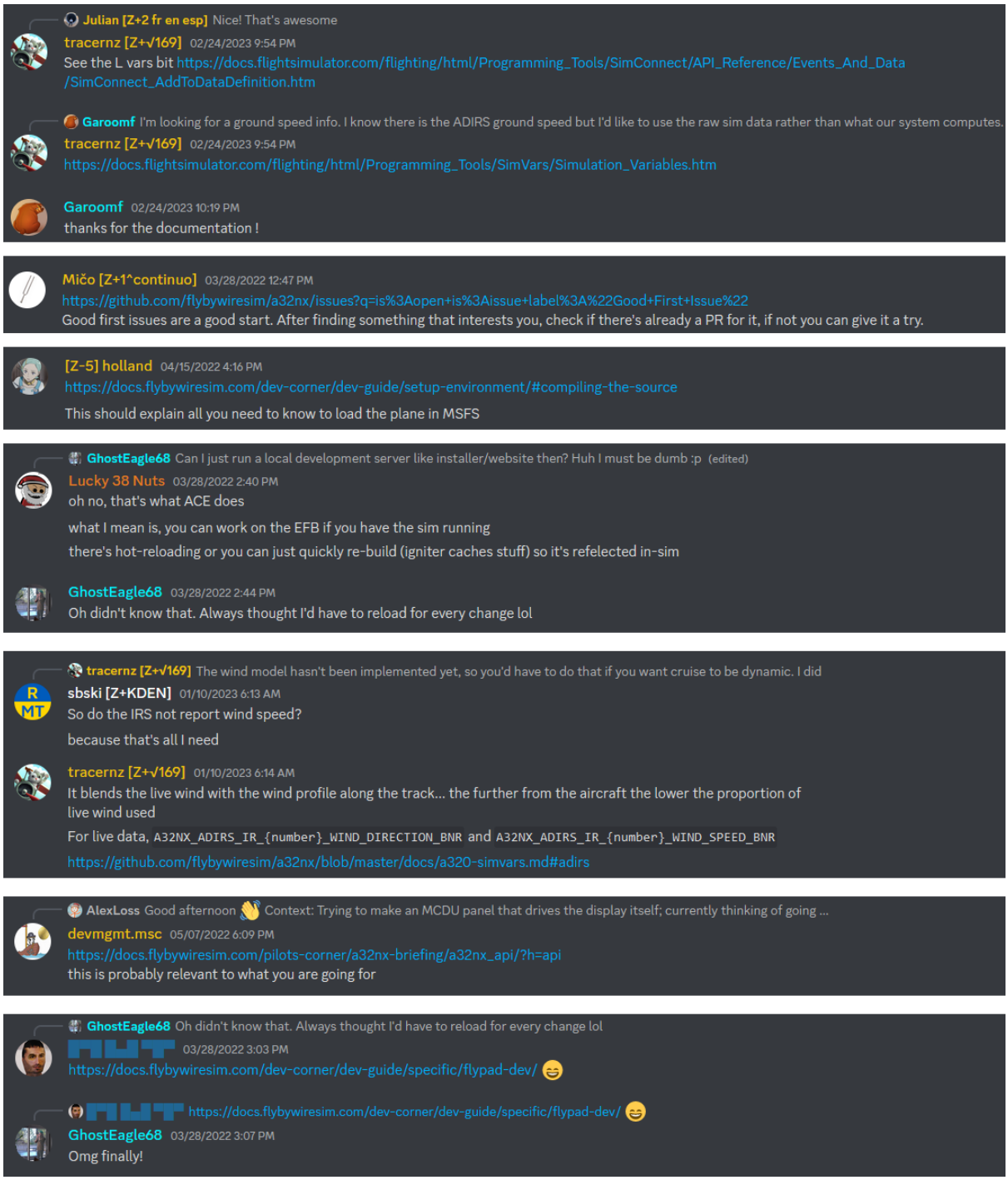

Figure 5.5. Questions answered with documentation

[7] `https://docs.flybywiresim.com/fbw-a32nx/faq/#aircraft`

Recalling the Research Question we sought to understand how does software documentation influence contributors motivation to participate in open-source communities.

Although the community invested significant effort in developing a comprehensive documentation, contributors remain reluctant to do their homework and consult the documentation materials available. Instead, they prefer to ask questions in the forums, questions that already have been asked as well as answered numerous times, and since have been documented down as well.

This is evidenced in the way questions are answered, using links to documentation instead of just being answered regularly. One member of the documentation team expressed their feelings on the docs channel as follows: *"So many questions in #a32nx-support we could easily answer with the beginner guide!! I'm tempted to send the preview link"*. To further support this claim, Figure 5.5 showcases a collection of chat fragments, question on the development channels that were answered with a link to the appropriate section of the project's documentation.

While documentation helps and make a difference, maintaining it is a tedious effort, especially as the codebase is ever growing and becoming more complex. Not finding accurate information however did not seem to deter the community developers from contributing. They managed to cope with the difficulties during the project's infancy, when the documentation was underdeveloped. They persevered driven by their passion for the product, and its community. They had good relationships with other community members, even forming friendships with a few of them. Through that, they could lay assured in a friendly atmosphere that they can ask for, as well as receive help and support whenever needed.

Overall, documentation matters! In the case of FlyByWire, it increased contributors' satisfaction and productivity not having to wait for answers. We have not noticed any uptick of popularity or correlation between the project's prevalence and the state of its documentation, however if nothing else, it helped trough freeing up more of the core contributors' time that they would otherwise have spent on writing up the selfsame answers that their fellow contributors asked for over and over again.

Threats to validity 6

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].

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.

Both the first and the second cycle coding were done by the authors independently, then the codes and themes identified were compared and discussed until reaching consensus. The observations were done in the same manner with the added change of only comparing the field notes summarizing them.

Here, we opted to classify and address them based on the way suggested and used by Yin [Yin, 2003] and Wohlin et al. [Wohlin et al., 2012] using internal and external validity:

6.1 Internal Validity

Internal validity is a way to ensure that the case design holds true, that the results of our study show a clear cause and effect relationship. The aim is to guarantee that the effect did not happen due to some unknown, unobserved or unexplained extraneous variables.

A key part of our research design was a single case study. One would argue that the scope was not broad enough or it did not have quantitative additions and in that regard they would be correct. On the other hand, we specifically used multiple qualitative research strategies and data collection methods combined such as interviews and observations striving for more in-depth understanding of this specific case strengthening it more instead of seeking generabilizity.

The number of interviews performed as per our research with the FlyByWire community

members was relatively small with seven (7) interviews performed total along the time span of two months, making our sample smaller than originally desired. However in regard to the data set, we still reached what we believe to be the point of saturation as defined by Saunders et al. [Saunders et al., 2018], as the thoughts and feelings reported by the participants as well as what we have discovered while observing the communities were consistent and in-line with each other. This in turn made repeatedly our repeatedly observed findings statistically more significant.

A large part of our findings are originating from what the interviewees said. It can be argued, that participants may have been reluctant to share their emotions. We found that this was not the case in the FlyByWire community. Interview participants shared their negative experiences as well on more than one occasions, telling us about more heated debates, or what they disdained with either the community or specific members, especially after their anonymity was being reassured.

Even though we had a single case in our study, it came to our attention very early during the gata gathering process that the the data presents a quite different state between the A32NX's documentation's early and current situation. While we considered its documentation as a positive example of how documentation should be done serving as a positive case, this discovery of deficient documentation being prevalent during the past of our case study inspired us to divide our single to serve as two cases to an extent: one negative and one positive. The negative case refers to the inadequate documentation present in the past, while the positive case highlights the current state of documentation.

Thus by incorporating both negative and positive cases or states serving as a base for comparisons between them, it strengthens our case study, overcoming the limitations of the initial plan of going with a single positive case. [Espedal et al., 2022] [Miles et al., 2013] I also reduces bias in the result while also helping us to understand and explain the original case better [Emigh, 1997] [Hanson, 2017].

6.2 External validity

External validity is the extent to which the findings and the conclusion from our study can be used or applied to other situations, cases and conditions, for example to other open-source projects. It answers the questions of generalizability and transferability. [Miles et al., 2013]

The findings of this study are about one select community, that is FlyByWire Simulations. Open-source communities however have inherently similar characteristics, generally focused on a specific or a set of specific projects as well as operate using a decentralized development model with contributors from all around the world at different timezones cooperating together to develop the software. In addition, as previously discussed, the motivation behind individuals

joining open-source projects showcases recurrent themes with the most prominent one being a shared passion for a hobby, topic or area of interest. Thus we believe that due to these inherent similarities, our findings would likely be applicable to other, similar open-source communities as well.

7 Discussion

7.1 Discussion

We have seen several research papers talking about contribution barriers that open-source contributors, especially newcomers face. Both Mendez et al. [Mendez et al., 2018] and Steinmacher et al. [Steinmacher et al., 2014a] identified general contribution barriers to newcomers, like social interaction issues, lack of technical knowledge, or problems with finding a way to start, a suitable first issue to try tackling.

We have also seen articles talking about some of them, mainly one specific family of these barriers that are various software documentation related issues, which we have also thoroughly categorized in Section 3.6. Aghajani et al. [Aghajani et al., 2019] enumerated and categorized an astonishing 162 different types of documentation issues, a few of which are: certain types of documentation artifacts like contribution guidelines, FAQ or build guide missing, documentation being fragmented into multiple platforms, or simply not being available.

Although while these studies exist, what none of them expands upon is the consequences of these issues. Will contributors drop out after being demoralised from these obstacles, or specifically documentation problems encountered, or will they persevere through finding out other ways to obtain the information that they sought? Our work shows that in certain communities new contributors may very well do the latter. In case of the FlyByWire community, during the earlier days of the project where the documentation was "extremely minimal" (P3), they still persisted through by coping mechanism such as making friends with the community, and help from their fellow members (see Section 5.2.5). These coping mechanism are different than the ones identified in other studies like the one done by Lee et al. [Lee et al., 2006]. Their research was performed in a corporate environment, they identified and were discussing factors like on-site labour organization, were focusing on costs as well as identified a centralised knowledge base and writing documentation itself a coping mechanism. The first two points are mostly extraneous in case of open-source projects, and writing documentation is inapplicable in our scenario, as we are focusing on the effects of inadequate documentation on new contributors, not having the domain knowledge to contribute to the knowledge base of a project. Overall, these points made it a worthwhile endeavour to look into these mechanism in open-source communities as well, in an antithetical situation where proper documentation was not existing before.

Practical implications for newcomer contributors is to not feel discouraged when initially not finding written help regarding an open-source project. Instead let passion prevail, and try to find information through other means. At the same time, we would suggest long-standing community members to try to build a community based on friendliness, warmth and sharing that encourages helping out their comrades when they ask for help.

Moreover, our research also explored something new, a phenomenon not yet documented. While there are existing studies on the evolution of open-source documentation like one done by Dagenais and Robillard [Dagenais and Robillard, 2010b]. They explored what kind of efforts and decisions go into writing documentation for open-source, how does the process look like as well as what parts of documentation do open-source contributors find the most useful. Their data source however consisted of interviews with developers from multiple popular open-source projects. In comparison, our focus is a single community, the FlyByWire A32nx team its transformation going from almost no documentation to excellent documentation. To the best of our knowledge, no previous study has investigated this transformation process in-depth. Nowadays, A32nx's documentation is there, and the documentation team is continuously trying to keep up with the changes. However, even though the documentation is present, we have noticed numerous new contributors asking questions, more specifically asking questions that have already been answered either in the "FAQ" page or the general documentation site of the project

Here, we would suggest new contributors to try to spend a little time exploring the project's documentation for potential answers for their questions first, before asking their fellow community members for help.

Another finding of ours was how prevalent issues relating to version control systems, mainly with Git and Github are, especially among those without formal software-related backgrounds, or students and fresh graduates from a relevant mayor. These findings are in line with the works of Isomöttönen and Cochez [Isomöttönen and Cochez, 2014], and Feliciano et al. [Feliciano et al., 2016] discussing that at many universities, Git is not part of the curricula, and even if it is, oftentimes it is not taught in a sufficient manner, hence students having problems with even easier concepts therein. To counter this issue, in the case of the FlyByWire community, documentation maintainers are actively discussing adding a general git introduction session to the core project documentation to help counter this issue. Even though we feel that the sentiment is nice, well needed and also welcome, but we would also argue that it is outside the scope of a specific project's documentation.

These findings hold implications for educators suggesting a need for inclusion as well as better incorporation of version control themes into university curricula as it is an invaluable skill both at open-source project and in software enterprises that many newcomers struggle with. Furthermore this in turn also have implications for students. The need for them to try to acquire git knowledge on their own, not neglect and sideline it in favour of core programming skills.

Conclusion 8

8.1 Conclusion

In conclusion, in this thesis we have explored how a projects' documentation and its quality can affect motivation of contributors to continue participating. First we presented relevant theories and literature, expanding upon problems frequently plaguing documentation. Then we showcased how open-source software plays a significant role in the modern technological landscape and how documentation can help with knowledge transfer within developer communities. However, related works have shown that barriers exist that can prevent contributors from effectively participating in open-source projects. A subset of these issues are related to documentation: Be it outdated, fragmented, too little, or too much; neither of them are as helpful as we would otherwise want them to be. Afterwards, we also detailed the methodologies to be used as part of our qualitative case study.

This prelude was followed by a presentation of our findings including taking threats to validity into account and the discussion of the project's contribution to the field.

As part of the findings it became evident that just as other factors, documentation and its state does not make or break whether a newcomer will continue contributing or not. In the FlyByWire community, we have seen how community members coped with deficient documentation in the project's infancy through making friends with their fellow community members and asking others for help. Future works could be about how different the results would be in other, less welcoming communities. There are multiple factors at play, but it is conceivable that in certain cases, it will be the project documentation's weight that might tip the scales in one direction or another.

We have illustrated, how following the FlyByWire documentation team turning around the documentation's state, certain members were still asking questions already answered in the documentation. On the other hand, more and more of these questions are being answered using links to parts of the documentation in lieu of standard responses.

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.

[Bianca M. Napoleão, 2020] Bianca M. Napoleão, Fabio Petrillo, S. H. (2020). Open source software development process: A systematic review. *Empirical Software Engineering*, pages 1–14.

[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, 2010a] Dagenais, B. and Robillard, M. P. (2010a). 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.

[Dagenais and Robillard, 2010b] Dagenais, B. and Robillard, M. P. (2010b). 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*, FSE '10, page 127–136, New York, NY, USA. Association for Computing Machinery.

[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*.

[Feliciano et al., 2016] Feliciano, J., Storey, M.-A., and Zagalsky, A. (2016). Student experiences using github in software engineering courses: A case study. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, page 422–431, New York, NY, USA. Association for Computing Machinery.

[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.

[Isomöttönen and Cochez, 2014] Isomöttönen, V. and Cochez, M. (2014). Challenges and confusions in learning version control with git. In Ermolayev, V., Mayr, H. C., Nikitchenko, M., Spivakovsky, A., and Zholtkevych, G., editors, *Information and Communication Technologies in Education, Research, and Industrial Applications*, pages 178–193, Cham. Springer International Publishing.

[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.

[Kelly, 2004] Kelly, J. (2004). Spirituality as a coping mechanism. *Dimensions of Critical Care Nursing*, 23(4):162–168.

[Kruglyk et al., 2020] Kruglyk, V., Bukreiev, D., Chornyi, P., Kupchak, E., and Sender, A. (2020). Discord platform as an online learning environment for emergencies. *Ukrainian Journal of Educational Studies and Information Technology*, 8(2):13–28.

[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.

[Lee et al., 2006] Lee, G., DeLone, W., and Espinosa, J. A. (2006). Ambidextrous coping strategies in globally distributed software development projects. *Communications of the ACM*, 49(10):35–40.

[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.

[Miles et al., 2013] Miles, M. B., Huberman, A. M., and Saldaña, J. (2013). *Qualitative Data Analysis: A Methods Sourcebook*. SAGE Publications, Inc, 3rd edition.

[Mock, 2019] Mock, K. (2019). Experiences using discord as platform for online tutoring and building a cs community. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 1284, New York, NY, USA. Association for Computing Machinery.

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

[Nicolas Anquetil, 2005] Nicolas Anquetil, K. M. d. O. (2005). A study of the documentation essential to software maintenance. *Empirical Software Engineering*, pages 1–9.

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

[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.

[Saunders et al., 2018] Saunders, B., Sim, J., Kingstone, T., Baker, S., Waterfield, J., Bartlam, B., Burroughs, H., and Jinks, C. (2018). Saturation in qualitative research: exploring its conceptualization and operationalization. *Quality Quantity*, 52.

[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.

[Sutton and Austin, 2015] Sutton, J. and Austin, Z. (2015). Qualitative research: Data collection, analysis, and management. *The Canadian journal of hospital pharmacy*, 68(3):226.

[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.

[Vikas Sitaram Chomal, 2014] Vikas Sitaram Chomal, J. R. S. (2014). Significance of software documentation in software development process. *International Journal of Engineering Innovation Research*, 3(8):8.

[Vladoiu and Constantinescu, 2020] Vladoiu, M. and Constantinescu, Z. (2020). Learning during covid-19 pandemic: Online education community, based on discord. In *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pages 1–6.

[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.