Factors Influencing Developer Adoption in Open-Source Projects : A Conceptual Framework

Ujunwa Njoku
ujunwanjoku90@gmail.com

December 5, 2025

Abstract

Open-source software (OSS) projects depend critically on attracting and retaining developers, but many projects struggle to build a strong, long-lasting community of contributors . This paper introduces the Developer Adoption Journey Model, which explains developer adoption as a series of stages instead of one single decision. The framework identifies five stages which include discovery, evaluation, first contribution, community Integration, and sustained engagement. By breaking down the steps developers go through when adopting a tool, the framework helps maintainers understand where developers get stuck and how to fix those issues. It also gives researchers a clear way to study the behavior of developers in open-source projects. This framework brings together recent research on why people accept new technology, what challenges newcomers face, and how communities work. By combining these ideas, it creates one model that better explains how developers start using and adopting open-source tools.

1 Introduction

Open-source software has become the backbone of modern technology infrastructure, powering everything from operating systems to artificial intelligence frameworks. The success of OSS projects fundamentally depends on their ability to attract and retain developer contributors [1]. However, many promising projects fail to build sustainable communities, with studies showing that the majority of OSS projects struggle to maintain active contributors beyond a small core team [20].

The challenge of developer adoption extends beyond simply making code publicly available. Projects face many challenges such as technical issues, community dynamics, and motivational factors. All of these influence the way developers discover, evaluate, contribute and stay involved long-term in a project. With millions of OSS projects on GitHub competing for attention, it's now essential to understand what influences developer adoption.

1.1 The Problem

Previous research has already found many important factors that affect how developers join and take part in open-source projects. Studies have examined barriers faced by

newcomers [19], motivations for contribution [1], and characteristics of successful projects [14]. However, these insights are spread out across different research areas, which makes it challenging to build a complete understanding of the adoption process.

Current models often view developer adoption as a yes-or-no choice, either a developer contributes or does not, while failing to recognize that adoption is a journey involving multiple steps, decisions and transitions along the way. A developer who discovers a project may never evaluate it, one who evaluates may never attempt a contribution, and one who contributes once may never return. Each transition point involves different factors, barriers, and decision-making criteria that existing frameworks do not adequately capture.

Furthermore, most existing frameworks focus on either organizational adoption of OSS or individual contributor motivations [12], but few bridge the gap between initial project discovery and long-term sustained engagement. This gap limits our ability to understand why some projects successfully convert casual users into active contributors, while others struggle to retain even interested developers.

1.2 Research Objectives

This paper addresses these limitations by proposing a comprehensive OSS conceptual framework that:

1. Understand developer adoption as a step-by-step journey rather than a single choice
2. Identifies specific factors and barriers relevant to each stage of the adoption process
3. Integrates insights from technology acceptance models, newcomer barrier research, and community participation studies
4. Provides actionable guidance for project maintainers seeking to improve developer adoption
5. Establishes a foundation for future empirical research on developer adoption patterns

1.3 Contributions

This work makes three primary contributions to OSS research and practice:

First, we introduce the Developer OSS Adoption Model, a framework that breaks the adoption process into five clear stages: Discovery, Evaluation, First Contribution Attempt, Community Integration, and Sustained Engagement. Looking at it in stages helps us understand how developers move from just knowing about a project to actively taking part in it.

Second, we combine research from several areas including: technology acceptance, newcomer onboarding, and community participation into a single framework. This approach uncovers patterns and relationships that are hidden when these areas are studied separately.

Third, we give practical advice for open-source project maintainers by showing specific actions they can take at each stage of the adoption process to reduce friction and increase the number of developers who fully adopt the project. These recommendations are based on understanding the particular challenges developers face at each stage, rather than on general best practices.

2 Background and Related Work

To understand developer adoption in open-source projects, we need to consider several theoretical ideas and research findings. This section reviews the key concepts that support our framework.

2.1 Technology Acceptance and Adoption Models

Technology acceptance models help us understand why people choose to adopt new technologies. The Technology Acceptance Model (TAM) suggests that the primary reasons people accept a technology are usefulness and ease of use. Recent research has modified these models to consider the special features of open-source software, such as voluntary participation and community-driven development [1].

The Diffusion of Innovations theory helps explain why people adopt new ideas or technologies. It identifies key characteristics that affect adoption, including relative advantage, compatibility, complexity, trialability, and observability. In open-source projects, these correspond to factors such as the benefits of performance, how well the project fits with existing workflows, the learning curve, the ability to try things out, and how visible the project's activity is [12].

However, models developed primarily to study technology adoption in organizations may not fully explain how people participate in open-source projects. Developers often adopt OSS projects not because their organization requires it, but to learn new skills, build their reputation, or support causes they care about. [14].

2.2 Barriers Faced by OSS Newcomers

A substantial body of research has shown the barriers that new contributors face when trying to work on open-source projects. These studies systematically identified obstacles in areas such as social interaction, documentation, contributing code, setting up the local environment, and finding suitable tasks [19].

Technical barriers include challenges with setting up development environments, understanding complex code-bases, and navigating unfamiliar tools and workflows [20]. Social barriers encompass fear of criticism, lack of response from maintainers, and difficulty establishing credibility in established communities [1].

Problems with documentation are common, and studies show that incomplete or outdated documentation makes it harder for newcomers to succeed [14]. Recent research also finds that having clear contribution guidelines, well-written architecture documentation, and beginner-friendly issues is strongly linked to successful first contributions [7].

2.3 Developer Motivation and Retention

Recent research shows that developers contribute to open-source projects for different reasons. Some are intrinsic, such as the chance to learn new skills, express creativity, or support the idea of free software [1]. Others are extrinsic, like advancing their career, gaining a good reputation, or solving technical problems they personally face [12].

Research shows that early positive experiences are very important for keeping developers engaged. Developers who get timely and constructive feedback on their first contributions are much more likely to continue contributing regularly [20]. On the other

hand, negative early experiences, such as harsh criticism or having contributions ignored, often lead to them quitting permanently [1].

Community factors are also important for keeping contributors involved. Projects with welcoming cultures, clear rules, and opportunities for contributors to take on more responsibility demonstrate higher retention rates [14]. Mentorship programs and responsive maintainers also help newcomers stay engaged [7].

2.4 Project Characteristics and Attractiveness

Research looking at individual projects has found certain features that affect whether developers adopt them. One important factor is project activity, which can be seen in how often commits are made and how quickly issues are addressed [12]. Projects with regular activity signal good health and engaged maintainers which tend to attract and retain contributors than those appearing dormant or abandoned.

Technical factors like the popularity of a programming language, the size of the project, and how modular its architecture is can influence adoption [7]. Developers often prefer projects using familiar technologies and those structured in ways they can contribute independently without needing a lot of coordination.

Social factors such as the project's reputation, the credibility of its maintainers, and the size of its community also influence how attractive it is [1]. Studies show that very large communities can discourage newcomers who feel they have few opportunities to contribute meaningfully, while very small communities may suggest that the project is not likely to succeed [14].

2.5 Gaps in Current Understanding

While existing research offers useful insights, there are still some gaps. First, most studies look at individual factors separately instead of examining how they work together throughout the adoption process. Second, the timing of adoption is not well understood, meaning how factors change in importance as developers move from first learning about a tool to regularly using it. Third, there are few frameworks that connect the initial decision to adopt a tool with long-term continued use.

This proposed Developer Adoption Journey Model addresses these gaps by providing a stage-based framework that combines technical, social, and individual factors and recognizes that the adoption process is constantly changing.

3 The Developer Adoption Journey Model

This conceptual framework models developer open-source adoption as a journey through five distinct stages. Each stage represents a transition point where developers decide whether to continue with a project. Identifying the factors and barriers at each stage allows for targeted interventions to improve adoption outcomes.

3.1 Framework Overview

The Developer Adoption Journey Model consists of five sequential stages:

1. **Discovery**: Becoming aware of the project and forming initial impressions

2. **Evaluation**: Systematically evaluating whether the project meets technical and personal needs
3. **First Contribution Attempt**: Overcoming barriers to make an initial contribution
4. **Community Integration**: Establishing social connections and credibility within the project community
5. **Sustained Engagement**: Maintaining long-term participation and potentially taking on leadership roles

While we present these stages sequentially, developers may iterate between stages, skip stages, or exit at any point. The framework is meant to help analyze developer behavior, not to dictate exactly how developers should act.

3.2 Stage 1: Discovery and Initial Attraction

3.2.1 Stage Characteristics

The discovery stage occurs when developers first become aware of a project's existence. This awareness may arise through various channels such as web searches for solutions to specific problems, recommendations from colleagues or online communities, social media exposure, conference presentations, or exploring platforms like GitHub.

Initial attraction is formed within seconds or minutes as developers make rapid judgments about whether a project warrants further investigation. These first impressions are based on limited information but play a major role in deciding whether they will explore the project further.

3.2.2 Key Factors

Project Visibility: Projects should be easy to find through relevant searches and visible on popular platforms. Their visibility depends on factors such as SEO optimization, GitHub stars and forks, social media presence, and mentions in technical blogs or related project documentation.

Perceived Usefulness: The project should solve a problem or meet a need that developers understand. The README and project description must clearly explain the purpose of the project and how it can be used so that its usefulness is easy to see.

First Impressions: Visual presentation, documentation quality and clear signs of activity such as recent commits and active issues shape initial perceptions. Professional presentation makes the project seem reliable and well-maintained.

Technology Stack Familiarity: Projects using technologies the developer already knows or wants to learn are more attractive. Unfamiliar or obscure technology choices create barriers at this early stage.

3.2.3 Common Barriers

Confusing or overwhelming documentation can make it hard for developers to quickly understand what the project does. Signals of abandonment such as lack of recent activity or unanswered issues cause developers to question project viability. Unclear licensing can also discourage developers who are concerned about legal issues.

3.2.4 Implications for Practice

Projects should invest in clear, concise README files that explain what the project does, its main features, and target users. They should also show visible activity so people know the project is still alive, even when no new features are added. Using good keywords, tags, and links from related projects helps more people find it. Having a clean and professional look, with a logo, screenshots, and example use cases, makes the project more appealing at first glance.

3.3 Stage 2: Evaluation and Assessment

3.3.1 Stage Characteristics

Developers who pass through discovery stage enter a more systematic evaluation phase. During this stage, they assess whether the project genuinely meets their needs and if it is worth spending time to learn and contribute. This evaluation may take minutes to days depending on project complexity and developer needs.

3.3.2 Key Factors

Documentation Completeness: Comprehensive documentation including installation guides, API references, architecture overviews, and usage examples enables thorough evaluation. Missing or outdated documentation creates uncertainty about project capabilities.

Code Quality and Architecture: Developers examine code structure, testing practices, and how the system is built to see if it's easy to maintain or difficult to contribute to. Clean, readable, and consistent code shows that the project is well-managed.

Learning Curve: The effort required to understand and use the project influences adoption decisions. A long learning curve can push them away, especially when they don't have much time or need a quick solution.

License Compatibility: Developers verify that project licenses align with their intended use cases, whether personal, academic, or commercial. License incompatibility is a dealbreaker for many developers.

Project Activity and Maintenance Status: Recent commits, frequent updates, and quick responses to issues show that a project is actively maintained. But if there are many old, unresolved issues or no updates for a long time, it may mean the project is no longer being maintained.

Community Size and Engagement: The number of contributors, the quality of the conversations, and how quickly maintainers reply show how healthy the project is and how likely you are to get help.

3.3.3 Common Barriers

When documentation is not enough, developers must figure out how things work by reading the code, which makes the tool harder to evaluate. If the code is messy or not well tested, developers worry that the project will be difficult to maintain. Licenses that are confusing or too restrictive make developers unsure about legal risks. When maintainers are stressed, slow to respond, or not active, developers lose confidence in the project and look for more reliable options.

3.3.4 Implications for Practice

Maintaining clear and updated documentation is one of the most important things at this stage. A project should explain both how to use the software and how it works inside for developers who want to contribute. Releasing updates often and responding quickly to issues shows that the project is actively maintained. When a project is open about how decisions are made, it helps developers trust that the project is stable.

3.4 Stage 3: First Contribution Attempt

3.4.1 Stage Characteristics

The first attempt to contribute marks an important step from being a passive user to an active contributor. This stage has many technical and social challenges, which often cause people to give up. Successfully completing this stage is a strong indicator of long-term involvement.

3.4.2 Key Factors

Clear Contribution Guidelines: Providing clear rules for coding style, testing, commit messages, and pull requests helps reduce confusion and mistakes.

Accessible Issues: Having well-labeled tasks like "good first issue" or "help wanted" gives newcomers clear starting points. Clearly defined tasks allow contributions even without deep knowledge of the project.

Easy Development Environment Setup: Setting up a local development environment should be simple. Complicated setups or missing instructions make first contributions harder.

Barriers to First Pull Request: The difficulty of making changes, running tests, and submitting pull requests affects success. Tools that catch common errors early can help contributors make progress faster.

Quality of Initial Feedback: Fast, constructive, and friendly feedback from maintainers improves the newcomer experience. Slow or dismissive responses can lead to contributors leaving the project.

3.4.3 Common Barriers

Technical barriers include difficulty setting up development environments, understanding build systems, running test suites, and learning unfamiliar version control workflows. Social barriers include fear of criticism, uncertainty about whether contributions are welcome, and not knowing where to start. Process barriers involve confusion about contribution steps, testing requirements, and review processes.

3.4.4 Implications for Practice

Projects should have clear, tested contribution guides that explain every step from setting up the environment to submitting a pull request. They should label issues that are good for newcomers so that beginners know where to start. Using automated tools like continuous integration, code formatting checks, and complete test suites helps reduce manual work and gives fast feedback. Maintainers should respond quickly and positively

to first contributions because early encouraging experiences are very important for keeping newcomers engaged.

3.5 Stage 4: Community Integration

3.5.1 Stage Characteristics

Developers who make their first successful contributions move into a stage of community integration. In this stage, they form social connections, gain credibility, learn the community's norms, and develop a sense of belonging as they move from being outsiders to insiders.

3.5.2 Key Factors

Maintainer Responsiveness: When maintainers respond quickly to contributions, questions, and suggestions, it shows they value contributors' time and effort. Regular engagement builds trust and encourages people to keep participating.

Community Culture: How people interact, handle disagreements, and make decisions affects contributor experiences. Friendly and respectful communities that include diverse perspectives make it easier for newcomers to feel welcome.

Feedback Quality and Mentorship: Helpful feedback that explains why things are done a certain way and teaches best practices helps contributors learn faster. Mentorship, whether formal or informal, provides guidance and social support.

Recognition and Appreciation: Acknowledgment of contributors through changelogs, contributor lists, or simple messages of gratitude encourages positive behavior and emotional investment.

Opportunities for Growth: Giving contributors harder tasks and chances to take on more responsibility, such as reviewing contributions or mentoring newcomers, motivates them to stay engaged.

3.5.3 Common Barriers

Unfriendly cultures with harsh criticism, dismissive attitudes, or cliquish behavior make it hard for people to feel included. When contributions are not recognized, it creates the impression that effort is not valued. Decision-making processes that are unclear and exclude newer contributors make them feel like outsiders. Without clear paths for advancement, people are less motivated to continue developing their skills.

3.5.4 Implications for Practice

Projects should have clear rules that encourage respectful and inclusive interactions. Maintainers should set a good example and step in when these standards are not followed. Recognizing contributions, such as using automated changelogs or highlighting contributors, helps people feel appreciated. Providing clear ways for contributors to take on more responsibility, like becoming a reviewer or a maintainer, encourages them to stay engaged.

3.6 Stage 5: Sustained Engagement

3.6.1 Stage Characteristics

Sustained engagement happens when a developer moves from being an occasional contributor to a core member of the community. At this stage, they understand the project's goals, have built strong social connections, and often take on leadership or mentorship roles. Keeping developers engaged depends on how well the project's direction matches their personal goals.

3.6.2 Key Factors

Skill Development Opportunities: Projects that let developers learn new technologies, gain expertise, and tackle challenging problems keep them interested by supporting their career growth and curiosity.

Alignment with Personal Goals: Developers stay engaged when the project matches their personal goals, whether that is learning, building a reputation, solving problems, or supporting a cause.

Social Connections and Belonging: Having strong relationships with other contributors and feeling part of a community motivates people to keep participating even if external rewards decrease.

Sense of Ownership: Developers who feel they have control over project direction and outcomes are more committed. Being able to make decisions and influence plans increases investment.

Work-Life Integration: The ability to contribute without interfering with other responsibilities helps sustain long-term participation. Projects that respect boundaries and avoid overwork create healthier communities.

3.6.3 Common Barriers

Projects that stagnate or change direction in ways that go against contributors' values reduce motivation. When maintainers experience burnout and fail to make decisions, active contributors become frustrated. Toxic behavior or unresolved conflicts weaken the community. If contributors face excessive time demands that cause burnout, they may abandon the project.

3.6.4 Implications for Practice

Projects should give contributors chances to grow and take on more responsibility. This helps them learn new skills and develop leadership. Governance should be clear and include input from contributors so everyone feels involved and responsible. Building a strong community culture with regular communication, shared routines like releases and retrospectives, and social interactions helps contributors feel a sense of belonging. Respecting their time and avoiding burnout by setting reasonable expectations and sharing responsibilities keeps the project sustainable.

4 Discussion

4.1 Theoretical Contributions

The Developer OSS Adoption Journey Model offers several important contributions to open-source research. It views adoption as a process with multiple stages instead of just a single decision. This approach helps us understand how developers move from first learning about a project to becoming long-term, active participants. It also shows that different factors are important at different stages, which means that efforts to support adoption should be designed to match each stage.

Second, the framework combines ideas from different research areas, including technology acceptance theory, studies on newcomer barriers, and community participation research, into a single model. By doing this, it shows patterns that are hidden when these areas are studied separately and offers a complete way to understand developer behavior.

Third, the model clearly considers changes over time and two-way relationships. Developers can move back and forth between stages, and factors that are important at one stage may not matter as much at another. This approach better captures the complexity of real-world adoption processes.

4.2 Practical Implications

For OSS project maintainers, the framework gives clear guidance on how to increase developer adoption. Instead of general best practices, it suggests specific actions for each stage:

At the Discovery stage, make the project easy to find and clearly explain its purpose. At the Evaluation stage, provide complete documentation and show that the project is actively maintained. At the First Contribution stage, reduce setup difficulties and give clear starting points. At the Community Integration stage, create a welcoming culture and offer mentorship. At the Sustained Engagement stage, provide opportunities for growth and let contributors share in ownership.

The framework also provides ways to measure how healthy the adoption process is. Projects can track how many developers move from one stage to the next to find where problems occur. For example, if many developers evaluate the project but few contribute, it may mean the contribution documentation is not clear. If few developers move from integrating the tool to staying engaged over time, it could point to cultural or community issues.

4.3 Implications for Future Research

The framework provides many opportunities for empirical research. Quantitative studies could test the stage model by examining large amounts of GitHub data to find common patterns and points where developers stop progressing. Longitudinal studies that follow individual developers could show which factors help them move successfully from one stage to the next.

Qualitative research can study developers' experiences at each stage and find obstacles and helpful factors that the framework does not show. Comparative case studies of projects with high adoption to those with low adoption can reveal practices at each stage that lead to success.

Experimental research could test ideas derived from the framework. For example, controlled studies could check if giving newcomers easy-to-start issues helps them make their first contributions, or if mentorship programs help them become part of the community more effectively.

4.4 Limitations

Several limitations should be noted. First, the framework is conceptual and has not been tested with real data. Although it is based on existing research, the specific stages and categories of factors still need to be validated empirically.

Second, the framework may make developer behavior seem too simple by suggesting a straight, step-by-step process. In reality, adoption journeys are more complex, with stages that can be skipped, repeated, or experienced differently by each individual.

Third, the framework mainly looks at individual developers and may give less attention to organizational or ecosystem-level factors. Developers working in companies face different challenges than independent contributors.

Fourth, the framework assumes that developers join OSS projects voluntarily and may not fully apply in situations where they are required to contribute as part of their job.

5 Conclusion

This paper introduces the Developer OSS Adoption Journey Model, a framework for understanding how developers find, assess, contribute to, and continue participating in open-source projects. The model breaks adoption into five stages: Discovery, Evaluation, First Contribution, Community Integration, and Sustained Engagement. This approach offers a clear way to study the factors that affect developer involvement.

The framework combines ideas from technology acceptance, research on barriers faced by newcomers, and studies of community participation to create a single model that fills gaps in current understanding. It shows that different factors are important at different stages and that successful adoption depends on addressing the specific barriers and enablers at each stage.

For practitioners, the framework gives clear guidance on how to improve developer adoption using specific interventions at each stage. For researchers, it serves as a foundation for future studies on developer behavior in open-source software ecosystems.

As open-source software continue to grow, it is important to understand and improve how developers adopt it. We hope this framework contribute to both the theory and practice of creating sustainable open-source communities.

6 Future Work

Future research should aim to test the framework using both large-scale quantitative studies and detailed qualitative investigations. Key priorities include:

- Validating the the five-stage structure by studying how developers progress in different open-source projects
- Identifying factors that help developers move successfully from one stage to the next

- Developing metrics and tools to evaluate the health of the adoption process
- Testing strategies suggested by the framework in controlled experiments
- Expanding the framework to include organizational and ecosystem-level influences
- Investigating cultural and demographic factors that affect the adoption processes

By exploring these areas, we can better understand developer adoption and provide practical guidance for building strong open-source communities.

References

[1] Balali, S., Steinmacher, I., Annamalai, U., Sarma, A., & Gerosa, M. A. (2018). Newcomers' barriers... Is that all? An analysis of mentors' and newcomers' barriers in OSS projects. *Computer Supported Cooperative Work*, 27, 679-714.

[2] Begel, A., Bosch, J., & Storey, M. A. (2013). Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. *IEEE Software*, 30(1), 52-66.

[3] Canfora, G., Di Penta, M., Oliveto, R., & Panichella, S. (2012). Who is going to mentor newcomers in open source projects? In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (pp. 1-4).

[4] Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2008). Free/Libre open-source software development: What we know and what we do not know. *ACM Computing Surveys*, 44(2), 1-35.

[5] Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3), 319-340.

[6] Dedrick, J., & West, J. (2008). Movement ideology vs. user pragmatism in the organizational adoption of open source software. In J. Feller et al. (Eds.), *Open Source Development, Communities and Quality* (pp. 23-34). Springer.

[7] Furtado, D., Paixão, K., & Maia, M. (2023). Do CONTRIBUTING files provide information about OSS newcomers' onboarding barriers? In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (pp. 1-6). IEEE.

[8] Hahn, J., Moon, J. Y., & Zhang, C. (2008). Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. *Information Systems Research*, 19(3), 369-391.

[9] Hars, A., & Ou, S. (2002). Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce*, 6(3), 25-39.

[10] Lakhani, K. R., & Wolf, R. G. (2005). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In J. Feller et al. (Eds.), *Perspectives on Free and Open Source Software* (pp. 3-22). MIT Press.

[11] Lee, G. K., & Cole, R. E. (2006). From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organization Science*, 17(6), 633-649.

[12] Lee, A., Carver, J. C., & Bosu, A. (2017). Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: A survey. In *Proceedings of the 39th International Conference on Software Engineering* (pp. 187-197).

[13] Midha, V., & Palvia, P. (2010). Factors affecting the success of open source software development. *ACM SIGMIS Database*, 41(2), 28-47.

[14] Pinto, G., Steinmacher, I., & Gerosa, M. A. (2019). More common than you think: An in-depth study of casual contributors. *IEEE Software*, 36(2), 44-58.

[15] Rogers, E. M. (2003). *Diffusion of Innovations* (5th ed.). Free Press.

[16] Shah, S. K. (2006). Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52(7), 1000-1014.

[17] Singh, V., & Twidale, M. B. (2012). Openness and the future of e-science. In *Annual Conference of the Computer Human Interaction Special Interest Group of the Human Factors and Ergonomics Society of Australia*.

[18] Steinmacher, I., Silva, M. A., Gerosa, M. A., & Redmiles, D. F. (2014). A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, 59, 67-85.

[19] Steinmacher, I., Graciotto Silva, M. A., Gerosa, M. A., & 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.

[20] Steinmacher, I., Conte, T., Gerosa, M. A., & Redmiles, D. F. (2018). The hard life of open source software project newcomers. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering* (pp. 72-78).

[21] Ven, K., Verelst, J., & Mannaert, H. (2008). Should you adopt open source software? *IEEE Software*, 25(3), 54-59.

[22] von Krogh, G., Spaeth, S., & Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7), 1217-1241.

[23] Zhou, M., & Mockus, A. (2012). What make long term contributors: Willingness and opportunity in OSS community. In *Proceedings of the 34th International Conference on Software Engineering* (pp. 518-528).