**Understanding Successful Mentor-Mentee Relationships in Open-Source Software Communities: Current Research**

Jasmine Mishra

University of British Columbia

Honours Thesis

**Introduction**

Understanding successful relationships between mentors and mentees can help us learn how to build stronger open-source software communities. My goal is to collect open-source software mentor-mentee data from available data sources and build a well curated dataset that will feed recommendation systems that pair mentors and mentees successfully. An understanding of the current literature and knowledge of mentorship relationships will aide me in my data mining.

My research questions:

What is the current knowledge on the success of mentorships in OSS?

What are the popular modes of communication in OSS?

What are the data collection methods used for existing research on OSS relationships?

What are the characteristics that make up a successful mentorship in these communities and how can this knowledge be used to better pair mentors and mentees?

I will focus on successful mentorship relationships in order to find data that will be useful for the creation of a recommendation system.

**Background**

Open-source software development communities are semi-organized groups of contributors who maintain, develop, and produce software. Some of the largest Open-source software include Linux, Apache Software Foundation, OpenStack, and various GitHub communities. Much of the development on open-source software is self-participation based, as the communities’ organisation are also often self-governed. This means that the acquisition and sustainment of newcomer developers (also known within the development community as newbies) into a community is crucial for its longevity.[[1]](#footnote-1) In order to smoothly onboard newcomers into a community, newcomers often work on designated starter projects for newbies and have mentors helping them adjust to the new environment.

One of the issues with onboarding new members into projects can be captured by Brook’s Law, which states that adding more manpower to a software project will add more schedule delays rather than prolong it. Although this law is an oversimplification of the issues regarding new team members, it does hold some truth. Adding new members to a project does have the potential to slow the team, are costly, and create additional learning, adding various overhead to team dynamics. This overhead can be decreased using mentorships, as members are able to more smoothly onboard into a project with the help of an experienced developer. Mentors will aid newbies with adjusting to common practices in the community including workflow, task management, coding, documentation, using new tools, and more.

**Mentorship Success**

In “Onboarding in Open-Source Projects”, it was found that mentored newbie developers had significantly more activities on GitHub than unmentored developers. We can see that the most activities were performed when the developers were working alongside their mentors. New developers build trust and gain deeper access to important project members more quickly, as they become more involved in the project. A problem with mentorship is that the mentor loses some of their normal duties while mentoring the newcomer. It was found that during the mentorship period, there is a significant drop in contributions. However, the newcomer has also contributed during this time of loss too and having lots of new members can gain even greater total contributions down the road. Mentorships for onboarding new members into an open-source software project helps with increase productivity, and management of all these new members.[[2]](#footnote-2)

MORE ABOUT WHY MENTORSHIPS ARE GOOD.

Newbies that submit pull requests that reference issues with a mentor present on that issue are more likely to say for their second year after their first year than newbies on a project without a mentor. Providing mentorship in early stages of development is seen to make newbies stay longer. [[3]](#footnote-3)

Implicit mentoring

Structured vs. unstructured communication.

**Modes of communication**

There are a variety of communication methods that mentors, and mentees use. The preferred method of communication depends on the standards and common practices of the specific Open-Source Software Community, and what the mentor and mentee personally agree on together. The most popular communication methods are email, Slack, Internet Relay Chat (decreasing in popularity), GitHub, task management boards like Jira, and mailing lists.

Apache Software Foundation projects follow the principles of open communication logging discussion online:

***“Open Communications:*** as a virtual organization, the ASF requires all communications related to code and decision-making to be publicly accessible to ensure asynchronous collaboration, as necessitated by a globally distributed community. Project mailing lists are archived, publicly accessible, and include:

* + dev@ (primary project development)
  + user@ (user community discussion and peer support)
  + commits@ (automated source change notifications)
  + occasionally supporting roles such as marketing@ (project visibility)”

Mailing lists are used amongst several of the largest open-source software communities. Teams using mailing lists for discussion of the project. They are also a helpful tool for recruiting new members. “Many online communities recruit among friends, for example, by mining current members’ email lists.” Recruiting among friends creates strong bonds in the community and more commitment, as people will participate with their friends there.

Mailing lists are also used by newcomers to open-source software development to find teams. Newcomers will send an email on a mailing list looking for a mentor or team that will accept them into their project. The first interactions that a newcomer has in an open-source software community is very important. There is a correlation with successful future participation of newcomers with timely responses to their initial mailing list email.

The most important part of mailing lists is visibility – mailing lists are public information that anyone can read and even mine/scrape.

Though the use of mailing lists is prevalent, development teams often require more than just one form of communication. Because of the documentation heavy nature of software, developers need to constantly read and review code directly in their development channels, such as GitHub.

Developers also use issue tracking management tools such as Jira to track tasks and allow project management. These tools can be linked to git, sharing and tracking management data further. Some communities, like Eclipse Open Source, have their own issue tracking management tools that are linked to their mailing list conversations and their GitHub. This linked communication is highly organized and can be used to track conversations among software developers across all their communication forms.

**Hackathons**

There are several existing hackathons that encourage newbie involvement, often among students, in Open-Source Software projects. Some of these include the more infamous Google Summer of Code and

The communication within hackathons is very similar to that of a newcomer approaching an Open-Source Software project on their own: “With the exception of the hackathon itself, all interactions between mentors and developers took place in the regular channels used in each OSS project, including mailing lists, discussion forums, blogs, social networks, and Internet relay chat.” Some hackathons are also purely online, meaning that the modes of communication are nearly identical.

**Data Collection Methods**

Determining how data has been collected on mentor-mentee relationships in the past can be used as inspiration for future projects. Much of the data on their relationships has been gathered using interviews and surveys from mentors and mentees. Other methods include gathering large amounts of data, conversational and statistical, from online sources and performing analysis.

In “Onboarding in Open-Source Projects”, the impact of mentorship was measured using GitHub activities (total amount of commits, pull requests, interactions by developers) to compare the activity of developers with and without mentors.

In “Association of mentors with Project success in the Apache Foundation Incubator” the association of the success of projects with the number of mentors on each project was measured by counting the amount of mentors on each project and compared it to the success of the project relative to all of the projects that were looked at in the study. Success was measured by amount of commits, gitstats, and overall project completion.

“The role of mentoring and project characteristic for onboarding in open source software projects” they focused on hackathons and student based organizations. Then they compared the success of projects and the amount of mentorship on said projects. They found that the more mature projects with developers that were well established tended to have a smoother onboarding process. They traced commits, project speed, and compared it to.

In “implicit mentoring: the unacknowledged developer efforts in open source software” conversation data was mined from Apache mailing lists and ‘implicit mentoring’ was watched – they used machine learning to filter the PR comments into different categories: instruction, suggestion, and mechanism to fix errors.

Overall, some of the largest sources to find conversational datasets in open-source software are mailing list archives and GitHub PR comments. One of the main issues of mining data on mentor mentee relationships is the difficulties in determining who is a mentor and who is a mentee without having to ask the community members directly.

**Preliminary Research**

Finding Mentors:

In order to find mentor-mentee pairs and their conversations, there are a few steps that can be taken manually. By looking at participants from the Google Summer of Code, it is easy to identify mentors and mentee. The first thing to do is find a project on the Google Summer of Code past projects website. The code for the project is linked right from Google Summer of Code. Then one can find the project’s mailing list conversation by looking at the project’s mailing list page on that open-source community’s website. All that is left is to sort through conversation data between the mentor and mentee and their code contributions to determine the nature of their relationship.

Analysis of mentor mentee pairs:

Google Summer of Code

* Case 1: One mentor with student mentees in different years
* Looking at mentee Sanatt Aboral from 2019 with mentor Philip Wenig
* PR #50 student introduces themself to the team and the part of the project that they want to work on - makes a proposal for how they would like to complete the project. Team members and the mentor discuss with the student some solutions to a possible issue with the project. The mentor has a stronger position/presence on the team to make suggestions and do approvals. July - Aug 2019
* Mentee has a forked repo that they work on and push issues from the fork to the main repo when approved by mentor
* Mentor gives mentee advice on their code and info about the current structure of the existing code
* Mentor tells mentee about Eclipse open-source rules regarding copyright
* Mentor seems more involved in giving the mentee precise advice

Case 2:  
-Himanshu Balasamanta with Philip Wenig 2020

- gives advice about how to close commits/use git: issue #133

-Mentee experience documented here on this website:

-<http://himanshubalasammanta.live/expirience/gsoc>

- mentee describes the mentor giving the mentee work to do and suggests things for them to learn. The mentee does not want to go against the advice of the mentor and does everything that they suggest to them

- this mentee continues to contribute to the project even after the summer of code has been completed - they are now top 2 contributor on the repo

- mentee is seen helping a new mentee later on! Mentee has become a team member on the project

- repo: <https://github.com/eclipse/swtchart>

Case 3:

- Manthan Raju Surkar with two mentors: awasum, Michael Vorburger

- uses GitHub and Apache Issues board - this issue board can be sorted by assignee or easy searching. The issues are linked back to git PRs

- awasum gives mentee suggested issues to work on - there is a list of issues that are tagged beginner/starter/newbie on their Apache issue board -

- Mentors give mentee long descriptions and help on how to code and why to do certain things - seen in PR #723

- there is a checklist system in their PRs, where the items on the checklist must be completed before the PR is merged - seems like a helpful tactic for newcomers to effectively complete tasks

-<https://gist.github.com/thesmallstar/265d4f75a3a91bedb6acbf54b3e67cf9>

-repo: <https://github.com/apache/fineract/pull/1299>

Observations among all three cases:

* Mailing list communication is similar to the code comments in GitHub, but more theory based and not directly about the specific lines code all the time.
* Mentee introduces themself to the project as a new contributor
* The mentee asks a question about the project and the mentor will answer most of the time here, almost never other team members.

There are some characteristics in these relationships that are noticeably different from when a mentor interacts without someone who is not a mentee and is just a community member. Some of these include that the mentor gives advice about community guidelines. They are more in-depth descriptions and learning when giving advice to mentees than to code reviewers. There are also resources provided just for newbies such as a mailing list that has sets of instructions for the mentee/newcomers to follow for the specific project and guidelines for the community.

**Conclusion**

Some ways that relationships can be categorized.

An existing mentorship recommendation system is from “who is going to mentor newcomers in open-source projects?” They mined data from OSS to see which developers would be well suited for mentorship. When a newcomer comes into the project, it finds a developer that has created many discussion posts/mailing list posts that are like the topic that the newcomer is wanting to learn about or work on. An issue with this software is that it does not have strict guidelines that help determine whether the senior developers are good mentors or not – it just pairs them with projects that are similar and that they are experienced in. While this is an important part of mentor-mentee pairing, the success of a mentorship also lies in whether the mentor is good at being a mentor.

Importance of proper pairing system – increased diversity in the open-source software community. Women and underrepresented POC groups are important to the development of software. Software development should not be dominated by a single group because it hinders inclusion, creativity, and more within the community and the actual software itself. Software created by only a certain group will have biased opinions, goals, designs, and more.

Better pairing mentors also means for the sustainment of diverse groups and a culture for inclusion within the actual open-source software community also.

**STUFF WE NEED**

- newbies vs. relationship with others

- how long do mentorships last?

- how newbies communicate with people in the project

- are there situations with lots of newbies and only a few mentoring

- more newbies clustered around a few mentors who are more helpful

**Things we need**

- identify newbies

- use one project

- 2021-2022

- who are the people who committed the first item

- what are the people that helped them – these are implicit mentors

PROPER DEFINITIONS

- who is a newbie

- who is a mentors

- find specific messages

- are all reviewers mentors?

How do these people act with newbies vs other projects

Amount of comments

Longer comments

Vs just code review

Sentiment – patience/nice – emotion

Do they seem nicer?

TO DO:  
- extract data from summer code

GOOGLE: Has the mentee been successful in the future?

What is it like for people who quit projects?

What is the difference between good relationships and bad relationships based on the success of the mentee?

Statistical differences between these things.

Is the good relationship because the person is a good mentor or is it because that person is just a good reviewer? Are they behaving this way with other coders too?

We want to see if mentor behaves with newbies different from other people

Is the relationship different?

**HAS ANYONE LOOKED AT:**  
What does it mean to have a successful relationships?

Is there already a systematic literature review on newcomers??

IGOR

**Mining Data Repositories Conference**

2023

1. **Analysis of Mentors and Newcomers Barriers in OSS Projects** [↑](#footnote-ref-1)
2. **“Onboarding in Open Source Projects”,** [↑](#footnote-ref-2)
3. **How Important is Mentoring for New Contributors in an OSS Project?** [↑](#footnote-ref-3)