Bibliographic information:

Fabian Fagerholm, Alejandro Sanchez Guinea, Jay Borenstein, Jürgen Münch. Onboarding in Open Source Projects. IEEE Software, 2014.

# Onboarding in Open Source Projects

**Fabian Fagerholm**
Department of Computer Science
University of Helsinki
P.O. Box 68
FI-00014 University of Helsinki
Finland
fabian.fagerholm@helsinki.fi

**Alejandro Sanchez Guinea**
Department of Computer Science
University of Helsinki
P.O. Box 68
FI-00014 University of Helsinki
Finland
azsanche@cs.helsinki.fi

**Jay Borenstein**
Department of Computer Science
Stanford University
353 Serra Mall
Stanford, CA 94305 USA
Facebook
1601 Willow Road
Menlo Park, CA 94025 USA
borenstein@cs.stanford.edu

**Jürgen Münch**
Department of Computer Science
University of Helsinki
P.O. Box 68
FI-00014 University of Helsinki
Finland
juergen.muench@cs.helsinki.fi

**Abstract**

In today's world, many companies turn to open source projects as a source for increased productivity and innovation. A major challenge with managing this kind of development is the onboarding of new developers into virtual teams which drive such projects. However, there is little guidance on how to arrange the initiation of new members into such teams and how to overcome the learning curve. This case study on Open Source Software projects shows that mentoring can have a significant impact on onboarding new members into virtual software development teams.

**Keywords**

onboarding; open source software projects; virtual teams; mentoring; global software development; distributed software development; case study

Imagine working for a software company that has decided to make one of its projects go open source. You expect to benefit from an influx of new talent, enabling you to grow your product beyond the capabilities of a single organization. Your continuous concern will be to involve new developers in your virtual team, but time is scarce and you risk slowing down progress by adding more people. In 1975, Fred Brooks observed that adding people to a late software project makes it later [1] – an observation that has become known as Brooks' Law. To this day, almost 40 years later, software project managers struggle with questions on how and when to introduce new people into software projects. Although Brooks acknowledged that the "law" is an "outrageous simplification", he observed two main factors that impede the introduction of newcomers: the ramp-up time it takes for them to learn enough about the technical content to become productive, and the increased communication overhead that results from more people having to coordinate work.

## The Need for Continuous Onboarding

The general problem of introducing new people into an existing organization is especially pronounced in distributed settings with virtual teams. Virtual teams are small, often temporary groups of knowledge workers, separated by geographical, temporal, and cultural distances which add to the communication challenges involved in getting them to work effectively [2]. Today, Open Source Software (OSS) projects are among the most distributed kinds of projects carried out by humans. They rely exclusively on virtual teams, whose members can be placed on a continuum ranging from more permanent core developers to more temporary peripheral developers. In many application domains, engagement with open source is an inevitable part of the computing business. To gain market share, companies may have no choice but to participate in an existing open source ecosystem. The possibilities for low-cost innovation, access to large-scale project capabilities, and opportunities for recruiting proven talent are among the factors driving companies towards open source. OSS also plays an important role in government IT and the open source approach is often considered an enabler for technology and knowledge transfer to developing countries. Simultaneously, open source development presents several challenges to organizations that are used to a more traditional workforce, with opaque organizational

boundaries, hierarchical management, and relatively long-term employment. The flexibility provided by virtual teams in open source projects requires rethinking resource management and integration of new project members.

Onboarding, or organizational socialization, is a process that helps newcomers become integrated members of their organization. As part of onboarding, new members learn the knowledge, skills, and behavior they need to succeed and be productive in their work [3]. Onboarding is well explored in the organizational management literature. In software engineering research, there are some case studies concerning the process of onboarding (e.g. [4, 5]). However, there is little research or advice on onboarding for open source projects with virtual teams. The literature does not provide evidence-based guidance that would help project managers successfully involve developers in such scenarios. In this article, we present findings that complement an earlier study on onboarding in open source projects [6, 7]. We previously showed the general effect of onboarding support on newcomer activity [6, 7] and the moderating effect of project characteristics, such as age, number of contributors, and appeal, on the speed of the onboarding process [7]. This article continues the analysis by examining developer activity during onboarding more closely, assessing the potential cost of mentoring in terms of lost productivity, and suggesting guidelines for using mentoring as an onboarding support mechanism. Rather than focusing on developer retention, we focus on the very initial stage of integrating newcomers, i.e. climbing the learning curve in virtual teams. This particular concern is especially relevant for virtual teams in an open source context, where developers join and leave at a rapid rate, and onboarding is needed on a continuous basis.

## ELEMENTS FOR ONBOARDING IN OPEN SOURCE

The precise practices and actions involved in onboarding differ depending on context. In this case, the context was a large-scale collaboration program with multiple universities and open source projects (see sidebar). The onboarding procedures consisted of two elements: a co-located Hackathon event, and mentoring by experienced open source developers. At a three-day Hackathon event at Facebook's headquarters in Palo Alto, California, student developer teams met face to face to start working on their respective OSS projects. A mentor from the project was assigned to each team. Three days of intensive coding and socialization allowed developers to

get to know each other and their mentors, and provided an immersive introduction to the world of distributed OSS development. Part of the activities of the Hackathon included familiarization with the code base, tools, and procedures used in the project.

Mentors were tasked with recommending and detailing tasks, explaining the software architecture, and assisting in technical development details. With the exception of the Hackathon, 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 (IRC).

Developers were free to work on any tasks relevant to their projects. Initially, mentors would typically direct them to small tasks suitable for novices. They were assumed to gradually start taking the initiative and tackle tasks of greater complexity. Most tasks involved programming, ranging from small bug fixes to complicated new features. Other tasks included writing test cases, creating new issues in tracking systems when new bugs were found, and improving some non-functional aspect of the software, taking into account maintainability, performance, and user experience.

The developers were integrated into each open source project and community through its regular procedures. They were exposed to the norms and implicit policies of each community. In addition, they received support from their mentors, from their local and remote team members, as well as any support provided by their home organizations. In company settings, similar support structures can realistically be enacted both to enable entry into external open source projects as well as to enable third parties to enter projects driven by the company itself.

## DOES MENTORING HELP?

To assess the impact of mentoring support on developers, we use a compound metric called activity. This is the sum of basic metrics that can be directly obtained from GitHub, a web-based hub for software development where most of the development took place. Thus, activity is defined here as the total number of commits, pull requests, and interactions by the developers considered.

A commit is a submission of source code changes to a version control system. A pull request is a notification that a proposed set of commits is available for integration into the main code branch. The changes can be reviewed and potential required modifications discussed before a final decision is made to integrate or reject the pull request. The amount of pull requests reflects the group work that occurs between developers, since they involve discussions and code reviews in which more than one developer is involved. Finally, we define an interaction as a single message posted by a developer in a GitHub discussion forum. Discussions are usually linked to commits or pull requests. They may concern source code modifications, messages justifying submitted modifications, or general messages related to the current development of a pull request or commit. The number of interactions corresponds to the amount of communication and group work between developers. Each component in the activity metric has an associated time stamp. This allows us to calculate the activity over a certain period of time for a certain set of developers.

We assess whether mentoring positively influences the performance of developers by comparing the activity of developers receiving mentoring support with the activity of developers that haven't received such support. The non-mentored group was randomly selected among developers who participate in the corresponding OSS project but are not part of the core development group, since being a core developer implies not only successful project involvement, but a level of expertise which cannot be expected from a newcomer.

To make a comparison over time, we defined a time-series sampling strategy. We sampled the weekly activity of each developer for 16 weeks. For the mentored group, we selected the initial week of the collaboration as the starting point. For the non-mentored group, we defined the starting point as one week before the first activity found for each developer. The weeks are thus relative to when the developer began their onboarding process.

Figure 1 shows the accumulated activity over time of developers with onboarding support compared to the non-supported developers. Initially, the progression of both groups is similar. However, as the onboarding process unfolds, activity among supported developers increases significantly compared to the activity registered by non-mentored developers. The level of activity in the supported group then continues to rise in a more or less constant fashion throughout the whole time period.

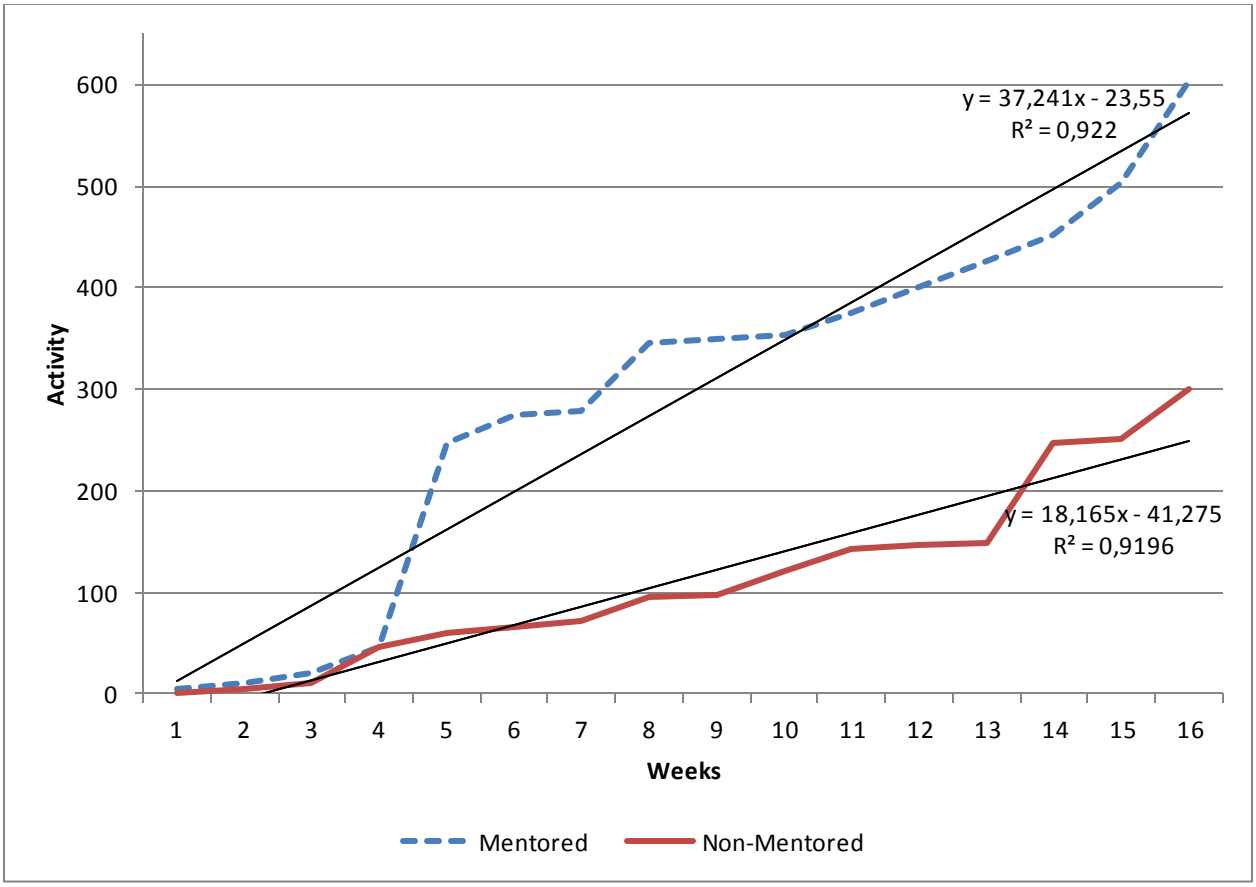


**Fig. 1. Accumulated activity over time of mentored and non-mentored developers. Mentored developers perform significantly higher than non-mentored developers. Performance bumps and plateaus likely reflect a human learning effect. The linear regression trendlines display a high goodness of fit ($R^2$) and illustrate the difference in activity over time more clearly.**

The bumps that appear in Figure 1 likely reflect the nature of human learning. When encountering a previously unknown task, we expect that developers engage in learning activities such as gathering and interpreting information to clarify the task and understand the software they are about to modify. Once they have accumulated enough knowledge and have reached a satisfactory level of understanding, they can begin to perform actual visible work.

The supported developers surpass the performance of the ones without support. We assume that the positive results with increased activity are related to the support given by the mentor throughout the duration of the program. Thus, the effect of mentoring is reflected as an increase in the activity and participation of a developer in an OSS project. For instance, if a mentor provides suggestions for suitable tasks that a developer could perform, this could have a positive impact on the number of commits by making developers focus on appropriate activities and reduce time wasted on other activities. Similarly, the amount of collaborative activities between developers could be affected by having the mentor mediate discussions and collaborative work. In our interpretation, the higher degree of activity among supported developers is, in part, a consequence of the mentoring support provided to the team.

## THE RELEVANCE OF INTERACTION

The aggregation of the activity metric can hide relevant patters which would be visible with more refined metrics. Such patterns can help understand the way in which developers contribute to their projects. For this reason, we analyze each component of activity separately.

Figure 2 shows the time series for each of the components of the activity metric for the developers that received onboarding support. The peaks reflect the learning effect shown in Figure 1. The figure shows that the largest portion of the activity metric stems from interactions with other developers, which seems natural for a group of newcomers who are still learning the processes and practices of a distributed project. Also, the interactions themselves may be valuable, as they may contain important information that spurs new ideas and innovation in the projects.

Since open source development relies on trust relationships built and maintained by developers themselves, it is important to prepare newcomers to interact according to the project culture. The same applies to other kinds of distributed projects with virtual teams. Project managers should emphasize the importance of informal communications, as it may increase the chances that new developers build trust and gain deeper access to important project members more quickly.

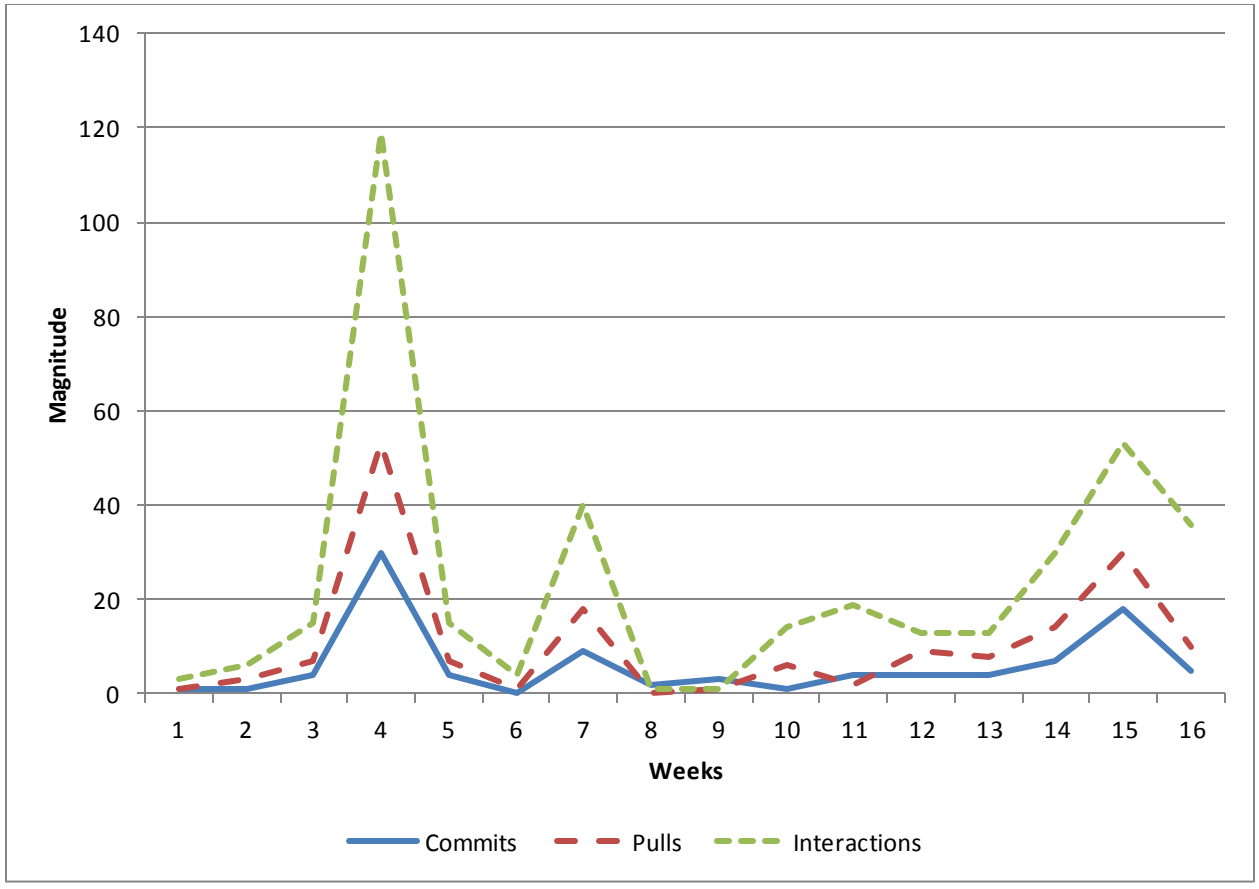


**Fig. 2. Detailed activity patterns of newcomers during the onboarding process. The amount of interactions reflects the coordination required to accomplish actual technical tasks.**

## DOES MENTORING PAY OFF?

It appears to be highly beneficial to utilize experienced developers to help newcomers become successfully involved in OSS projects. However, a potential problem may arise: the mentor needs to be engaged in support activities for some time, during which regular duties may suffer. It is important to consider the impact of the mentoring task on the performance of the mentor.

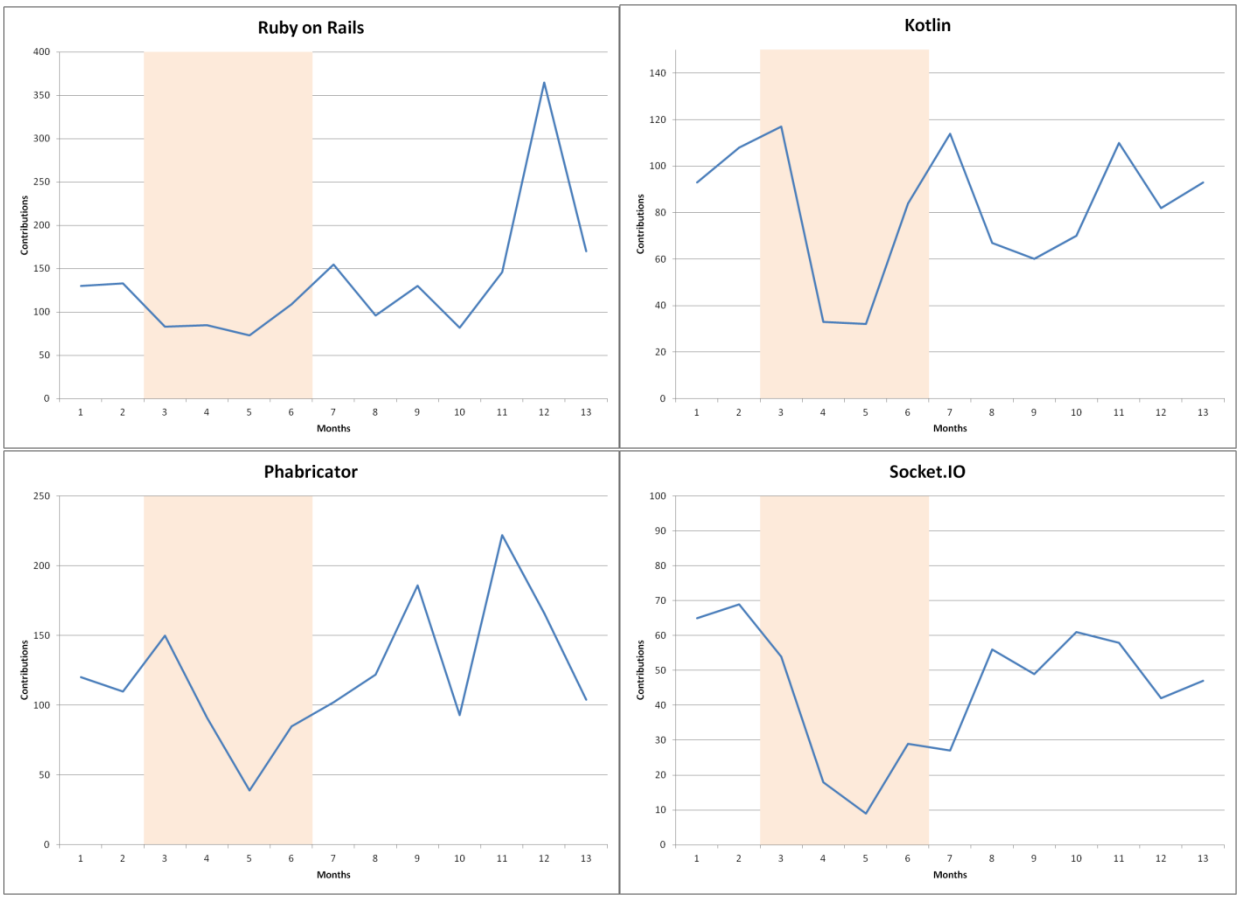


**Fig. 3. Mentor contributions over a time span of 13 months. Mentors are visibly less productive while engaged in mentoring tasks during the Open Academy Program (months three to six).**

To gain a coarse-grained understanding of the amount of work involved in mentoring besides other duties, we observed mentors' own development contributions to their respective OSS project over time. Figure 3 shows the development contributions by each mentor during a period of 13 months. The highlighted areas indicate the Open Academy program. Regardless of the differences in contribution patterns between each mentor, the amount of contributions is reduced while they are performing mentoring duties. This difference is seen more clearly in Figure 4, which compares the mentors' average contribution per week while performing mentoring tasks ("during mentoring") and while performing regularly ("regular development"). However, in all

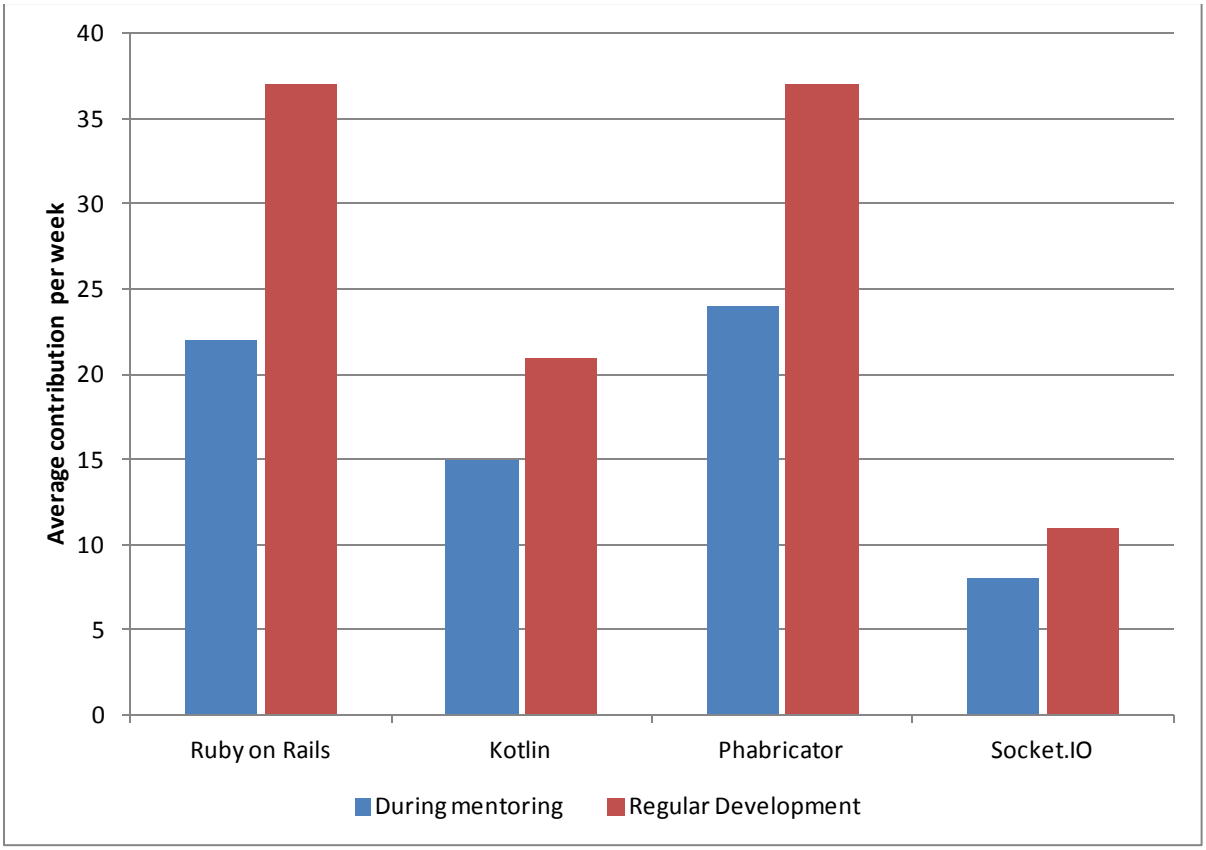


**Fig. 4. Comparison of average mentor contribution per week during mentoring and non-mentoring periods. Mentoring activities reduce mentor productivity, but contributions do continue even during the mentoring period.**

cases, mentors continued to contribute to their projects throughout the program. While the performance drop can be significant, it can be justified by the fact that it is limited to the onboarding period and its potential impact on increased productivity and increased innovation that may be gained from new project members. In practice, this opportunity cost must be evaluated on a case-by case basis. In many situations, the cost of mentoring is overshadowed by the potential benefits of gaining new members, even if they are temporary.

The onboarding process in the Open Academy program resulted in better performance than what would have been expected from the usual joining procedures in the participating OSS projects. Our interpretation is that the support given to developers did influence their onboarding process, allowing them to become more active. The results indicate that mentoring increases the chance of developers being exposed to, selecting, and performing tasks in a proactive and self-directed manner in open source projects. We assume that this applies in other kinds of settings where development is conducted by virtual teams.

The results presented here will inevitably vary depending on the particular context in which projects take place and depend on validity limitations of the study. Onboarding is a complex construct with multiple impact and context factors that may limit the applicability of the results. The analysis presented here cannot distinguish between mentoring and other onboarding support factors. Other contextual factors may play a role. For example, developers in the group receiving onboarding support may be more involved and spend more time more regularly with the project. Also, the Hackathon event may be another factor impacting the results. A further limitation is that we have not examined whether developers in the non-supported group have received any treatment during the observed period of time that would impact the results.

An interesting question is whether the effect of onboarding support is permanent after mentoring activities are removed. Since our focus was on newcomer initiation and not retention, and on rather temporary virtual teams, we did not examine this issue further. Future work could assess what is required to increase the likelihood of retaining project members after onboarding in highly volatile team environments.

In summary, mentoring is a viable support activity for onboarding, helping to rapidly integrate new members into OSS projects. Directly engaging with an open source project through an experienced mentor gives better results than having no mentor. Furthermore, improved onboarding performance seems to justify the cost of mentoring in many cases.

Our results have direct relevance for leading and managing virtual teams in an open source context. In light of our results, we have the following recommendations for project leaders and managers.

- Identify core developers who can spend a limited time on intensive mentoring. Provide direct incentives for mentoring. For example, the opportunity to get help for pending tasks can be attractive for potential mentors. Clearly limiting the duration of mentoring reduces the negative effect on the mentor's performance in other project tasks and can reduce some of the resistance to participate.
- Organize or sponsor collocated events, such as Hackathons, and use them to kick off the mentoring period. Face-to-face events can help team members and mentors to focus on problems which are difficult to overcome in a distributed setting, and can further boost the success of onboarding new members into virtual teams. Many open source projects already arrange periodic collocated events and welcome participation by newcomers. Engaging with these provides direct access to the project community.
- Expect considerable variation in performance increases over time. Assessing the cost and outcomes of mentoring requires understanding onboarding as a learning process which does not proceed linearly. Some onboarding activity will not be publicly visible. Engage directly with mentors and newcomers to gain insight of how onboarding is progressing.
- Adapt the onboarding program to project characteristics and culture and be prepared to provide different kinds of support to mentors in different kinds of projects. Take the maturity of the target project and its existing onboarding practices into account. Low-maturity projects may require more support to instill a productive mentoring culture, while mature projects may already have an existing culture of integrating new developers and may be ready for tailoring towards more specific inclusion targets. Consider taking on an expert with knowledge of open source projects and software engineering pedagogy to guide the mentors so that their expertise is transferred most effectively.

## REFERENCES

1. Frederick P. Brooks, Jr. “The Mythical Man-Month”. Addison-Wesley, 1975.

2. Nader, A. E., Shamsuddin, A., Zahari, T. “Virtual R & D teams in small and medium enterprises: A literature review”. Scientific Research and Essays, pp. 1575-1590, Vol. 4, No. 13, 2009.

3. Bauer, T. N. Erdogan, B. “Organizational socialization: The effective onboarding of new employees” in S. Zedeck (Ed.), APA Handbook of industrial and organizational psychology, Vol. 3, pp. 51-64. Washington, DC, USA, 2011, American Psychological Association.

4. Steinmacher, I., Wiese, I., Chaves, A.P., Gerosa, M.A., “Why do newcomers abandon open source software projects?,” 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 25-32, 2013.

5. A. Begel and B. Simon, “Novice Software Developers, All Over Again,” in Proceedings of the Fourth International Workshop on Computing Education Research (ICER), pp. 3-14, New York, 2008, ACM.

6. F. Fagerholm, Johnson, P., Sanchez Guinea, A., Borenstein, J., Münch, J., “Onboarding in Open Source Software Projects: A Preliminary Analysis,” IEEE 8th International Conference on Global Software Engineering Workshops (ICGSEW), 2013.

7. Fagerholm, F., Sanchez Guinea A., Münch, J., Borenstein, J. “The Role of Mentoring and Project Characteristics for Onboarding in Open Source Software Projects”. Empirical Software Engineering and Measurement (ESEM), 2014.

8. F. Fagerholm, Oza, N., Münch, J., “A Platform for Teaching Applied Distributed Software Development: The Ongoing Journey of the Helsinki Software Factory”, 3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development (CTGDSD), 2013.

## ABOUT THE AUTHORS

Fabian Fagerholm is a doctoral student at the University of Helsinki, working for the Department of Computer Science in its Software Systems Engineering Research group. He has driven the design, implementation, and operation of the department's Software Factory laboratory for experimental software engineering research and education. His main interests are human aspects of software engineering and software development processes. Fagerholm obtained his Master's degree from the University of Helsinki. Contact him at fabian.fagerholm@helsinki.fi.

Alejandro Sanchez Guinea is a research assistant at the University of Helsinki, working for the Department of Computer Science in its Software Systems Engineering Research group. His main research interests are software measurement, software processes improvement, and software engineering design methodologies. He obtained his Master's degree from the Institut Supérieur de l'Aéronautique et de l'Espace. Contact him at azsanche@cs.helsinki.fi.

Jay Borenstein teaches computer science at Stanford University and also runs Facebook Open Academy as part of a larger Facebook effort to modernize education. Jay finds it very fulfilling to help motivated, bright minds grow and succeed. Contact him at borenstein@cs.stanford.edu or through www.facebook.com/openacademyprogram.

Jürgen Münch is a professor of software systems engineering at the University of Helsinki, head of its Software Systems Engineering Research group, and principal investigator of the experimental R&D laboratory "Software Factory". His main interests are quantitative modeling and analysis of software systems and processes, data analytics, and continuous experimentation. He received his PhD degree (Dr. rer. nat.) in Computer Science from the University of Kaiserslautern, Germany. Contact him at juergen.muench@cs.helsinki.fi.

[sidebar]

## GLOBAL OPEN SOURCE PROGRAM

Starting in spring 2013, Stanford University and Facebook, Inc. launched the Open Academy program (www.facebook.com/openacademyprogram), which involves several open source projects and a significant number of top universities around the world. The intention of the program is to improve computer science university curricula through a practical and applied learning experience.

**Table 1. Participating Open Source Projects.**

| Project Name | Web Site | Included in This Study? |
|---|---|---|
| Freeseer | http://freeseer.github.io/ | No |
| Kotlin | http://kotlin.jetbrains.org/ | Yes |
| MongoDB | http://www.mongodb.org/ | No |
| Mozilla OpenBadges | http://openbadges.org/ | No |
| ReviewBoard | http://www.reviewboard.org/ | No |
| Phabricator | http://phabricator.org/ | Yes |
| PouchDB | http://pouchdb.com/ | No |
| Ruby on Rails | http://rubyonrails.org/ | Yes |
| Socket.IO | http://socket.io/ | Yes |

The findings we report here are based on results from the 2013 edition of the Open Academy program, including nine OSS projects, more than a dozen universities, and more than 120 students, referred to in the study as developers. Table 1 shows all projects participating in the program and indicates four projects which we examined in this study. Students at the University of Helsinki participated in these four projects through Software Factory, an experimental research and development laboratory [8], which allowed us to follow the onboarding process in these projects very closely. The projects represent a range of different project sizes, ages, and technology types.

The teams participating in the program followed the typical practices of open source development and can be characterized as virtual teams with members being distributed across

different geographical locations with their local cultures. All the participating open source projects had a wide temporal distribution, each receiving a constant stream of contributions around the clock.