Training the Future Workforce through Task Curation in an OSS Ecosystem

Anita Sarma, Rafael Leano
Oregon State University
Corvallis, OR, USA
{anita.sarma, leanor}
@oregonstate.edu

Marco Aurélio Gerosa
Institute of Mathematics and Statistics
University of São Paulo
São Paulo, SP, Brazil
gerosa@ime.usp.br

Igor Steinmacher
Department of Computing
Federal University of Technology –
Paraná – Campo Mourão-PR, Brazil
igorfs@utfpr.edu.br

ABSTRACT

Volunteers to Open Source Software (OSS) projects contribute not only to help creating software that they use, but also to gain skills and enrich their expertise and resumes. However, newcomers to OSS face several challenges when joining a project. Particularly, they do not know where to start, or choose tasks that they can be successful at. Here, we describe our vision towards BugExchange, a system that curates tasks from OSS projects and helps train newcomers. While evaluating and executing these tasks, newcomers can gain an understanding about the project, its technology, and concepts. There are many challenges in designing such a system. For example, identifying the information needs of newcomers, creating task recommendations that match newcomers' skills and career goals, and providing mentoring and networking support. We plan to leverage our previous work to conceive and prototype our system, which will include multiple research lines. BugExchange has the potential to improve newcomer learning experiences, reduce dropouts, and foster community building.

CCS Concepts

• Software and its engineering → Collaboration in software development → open source model

Keywords

Newcomers; onboarding; open source projects; task labeling.

1. INTRODUCTION

Open Source Software (OSS) projects provide a large variety of systems that are popular and extensively used [9]. Supporting newcomers to OSS projects is important not only for the survival, long-term success, and continuity of the open source contribution model, but also to create a workforce that has hands-on experience in programming [15]. Many volunteers to OSS projects contribute to enrich their expertise and resumes [6, 14].

However, as evidenced by previous work, newcomers face several barriers to place their first contribution to OSS projects [17, 23]: they need to find the right project to start contributing to, find open tasks that match their expertise, gather relevant information, and get support from project members in order to be successful in making a contribution. During this multi-step process, some newcomers may lose motivation and even give up contributing, leading to a low retention rate [19]. This low retention rate is concerning some communities, like Mozilla, Gnome and Apache, which provide special programs and strategies to attract, mentor and retain new developers.

Currently, there is no support in helping newcomers identify tasks (already reported issues, such as bug fixing, new features, i18n translations) that they can contribute to and learn from their peers, as they move forward in their path to become contributors. Further, while many projects are related, it is unknown whether, and to what extent, developers migrate, and hence, become newcomers to another project. Our prior work [8] has found that developers migrate across projects within an ecosystem, but interpreters and documenters are the most likely to gain from the benefit of knowledge that is transferrable across projects.

Our vision is to create a support structure for helping newcomers familiarize with technical and social aspects of an ecosystem of projects, generating a workforce of contributors who can transfer knowledge across projects. We envision an approach, where tasks collected from multiple projects are labeled and categorized based on the skills required to complete them, which can then be used to scaffold the learning for newcomers about specific skills, as well as concepts within a project. Another key idea is to create support structures of near-peer mentors, such that newcomers can learn from each other, as well as those individuals who have recently made the journey.

There are several challenges in this process. First, projects have tasks of different complexity, and require different skill sets. The skills, in this context, can be of different types: knowledge about specific programming constructs (e.g., inheritance, lists), language constructs (e.g., java annotations, java enums), frameworks (e.g., Laravel, cmocka, Spring, rails), or project components (e.g., a particular view implementation, or a specific protocol). For newcomers, it is difficult to assess these skill sets by themselves [18], while project contributors are short on time to annotate the tasks. Second, there is not a smooth learning trajectory comprising of tasks at increasing levels of complexity in a single project. Moreover, it is possible that there may not be any "simple" starter tasks available for the completely uninitiated. Third, a key way to learn about a project is through mentors [4]. However, newcomers are often shy of asking for help from core contributors [17], who in turn are busy, and do not have the time to mentor. Most newcomers are simply pointed to the project documentation and mailing list. Finally, it is an open question, whether contributors in one project can transfer their knowledge to another project in an ecosystem.

In summary, our envisioned system, BugExchange, will help newcomers onboard a new project, train a workforce that can participate in open source development, and facilitate support structures such that newcomers and contributors can learn from each other. Expanding our approach to encompass multiple projects in an ecosystem will help contributors create a richer portfolio, as well as help projects increase diversity and cross-pollination of ideas. We will realize our approach (BugExchange) by building on our prior work – FLOSScoach [16], a web portal that supports the first contributions of newcomers to OSS projects, and a study [11] in which we evaluated how well newcomers identify the skills required to complete a task in an OSS project.

2. BACKGROUND

When newcomers join an OSS project, they face many barriers that hinder their first contribution, leading in many cases to their dropping out [17]. Researchers have tried to understand the barriers that influence the retention of newcomers. Zhou and Mockus [24] identified newcomers who are more likely to remain in the project, so as to offer active support to them to enable them to become long-term contributors. Jensen et al. [7] analyzed whether emails sent by newcomers are quickly answered, whether gender and nationality influence the kind of answer received, and whether the reception of newcomers is different among users' and developers' lists. Park and Jensen [12] showed that visualization tools support the first steps of newcomers in an OSS project, helping them to find information more quickly.

Steinmacher et al. [17] proposed a model to help identify and better understand the barriers faced by newcomers. One of the most recurrent and relevant barriers was finding a task to start with. In an in-depth, follow-up study [18], they found that newcomers need additional information about the tasks or need support from the community to decide a task that is suitable for them. Other studies have also found that identifying an appropriate task is a key problem, since new developers have difficulty in finding bugs or features that are of interest, match their skill sets, are not duplicates, and are important to the community [21]. Similarly, Park and Jensen [12] reported that information about tasks that can be performed by newcomers is important.

There are few research works that have proposed approaches to deal with this problem. Čubranić et al. [2] presented a tool that recommends source code, emails messages, and bug reports to support newcomers. Another tool, Tesseract [21], enabled newcomers to identify similar bugs through synonym-based search. Although these tools can help newcomers by increasing their knowledge about the tasks and their complexity, there is still not enough support in helping newcomers identify appropriate tasks as they move forward in their path to become contributors.

3. VISION

Our approach involves: (1) mining open source project repositories to identify OSS ecosystems, and collect open tasks, (2) leveraging the (crowd) work produced by newcomers and contributors for cataloging tasks with the required skills to complete a task, (3) performing source code analysis to categorize tasks based on their complexities, and (4) using community building and incentive structures to create and foster a community of near-peer mentors.

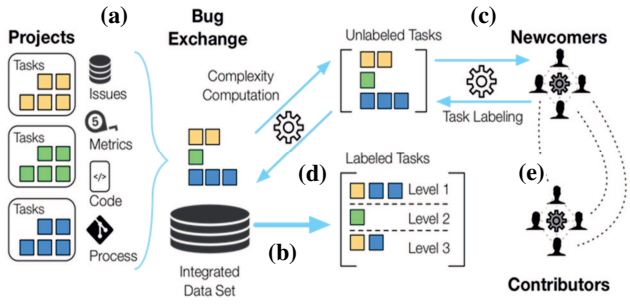

Figure 1. BugExchange Workflow.

More specifically, BugExchange (see Fig. 1) is a socio-technical approach that:

- Creates a clearing house of "open" tasks from multiple projects within a specific ecosystem – projects that have similar goals and share technical dependencies – providing a set of tasks that newcomers at different levels can attempt. To do so, we will collect different types of information (e.g., issues, work items, source code, code metrics, development processes) from open source projects (Fig. 1(a)), which will be cross-linked into a unified, integrated repository (Fig. 1(b)).
- Catalogs the skills required to complete a task, such that newcomers, as well as, contributors can assess whether they have the requisite skills to complete a task. We will explore different techniques, including an approach where newcomers label the skills that they perceive as required to perform the task (Fig. 1(c)) [11]. These starter tasks can allow a newcomer to familiarize with the project, its documentation and structure. We will also investigate how the crowd work compares to automated approaches (topic modeling) when identifying programming topics in tasks.
- Classifies tasks at different levels of complexity, such that contributors to a project can judge whether they have the competency to implement a task at a given complexity level (Fig. 1(d)). We will explore different approaches to calculating complexity metrics (e.g., cyclomatic complexity, centrality of a file, lines of code, or program structure).
- Recommends tasks to developers by matching their expertise and the skill level required to complete a task. We will explore different approaches, where we allow a newcomer to specify a skill that they want to learn or follow the development trajectory of a contributor in a project. We will also leverage developers' expertise recommendation approaches (e.g. [22]) to assess developers skills, to match newcomers and tasks.
- Provides a network of near-peer mentors, who are either at the same skill competency level or at a level above, who can support and learn from each other (Fig. 1(e)). Having near-peer mentors help in two ways. First, the questions and problems that newcomers face might be items that the near-peer mentors have recently experienced, and, therefore can provide guidance on. Second, it might help the near-peer mentor showcase their motivation and passion in becoming part of the community.

4. OPPORTUNITIES AND CHALLENGES

There are several challenges and research opportunities in creating a socio-technical approach, such as BugExchange, where automated techniques and humans operate side-by-side; these include:

Identifying required skills for a task: One of the key issues faced by newcomers is the (technical) learning curve. Newcomers have to understand the task requirements and the project itself to determine whether they can implement a task. While some issue trackers tag priorities or the type of an issue, none include the skills required to complete the task. It is an open question whether automated techniques, such as topic modeling, can identify required skills, or whether humans (or experts) are needed.

Determining task complexity: Simply identifying the skills may not be enough, as some tasks are more complex than others. For example, tasks that involve multiple files or files that are core to the system are harder to implement. Therefore, we need to compute complexity metrics for tasks, such that tasks can be categorized into varying difficulty levels. Many different mechanisms of computing complexity exist, we need to identify the approaches that are light-weight and work best in our context.

Identifying information needs and providing the documentation necessary to accomplish a task: A large part of the barriers that we identified in our previous work [17] could be mitigated by providing the appropriate documentation to the newcomers. While the recent rise of social media use by software developers has led to a plethora of documentation being available online for virtually any software product [20], this documentation is often poorly structured, and spread across the API official documentation,

blogs, forums, mailing lists, and other social media sites [13]. An open challenge is how to map the characteristics of a task to information needs, and then to automate the identification, extraction, summarization, and presentation of relevant documentation. We are already investigating an extended FLOSScoach that leverages existing and novel natural language processing techniques to automatically parse documentation. An additional challenge will be to map tasks to information needs, and them to documentation available internally and externally to the project.

Recommending tasks: Once we identify the skills required for a task and its complexity, we can match that with the expertise of a (newcomer) contributor. For the completely uninitiated, a starter task can be labeling the skills required for an open issue (task). This will require newcomers to read the issues, project documentation, the source code and its structure, helping them become familiar with the project. An open challenge is how to evaluate the quality of the labels (work) produced by the newcomers.

Once the system includes tasks that are labeled and the past history of contributors, it can recommend appropriate tasks to contributors. For example, one option is to provide tasks that require higher competence of a particular skill. Another option is to provide task recommendations based on a path followed by other developers. While, such scaffolding in task recommendations can help retain newcomers, it is challenging to execute. Creating recommendations requires not only knowing the background of contributors, but also taking into account aspects such as developers' motivation and availability, and the availability of a large number of closed and open, labeled tasks in the project.

Forming and sustaining peer mentor networks: Communication and mentoring are key in onboarding newcomers to a project. However, experts are busy with very limited time. Therefore, to be successful, BugExchange has to provide incentives to the project, and/or the contributors to be mentors. We conjecture that connecting newcomers to near-peer mentors – developers who have recently onboarded – is more beneficial since newcomers can learn about the challenges and processes from someone who has recently gone through the process, as pointed out by Glassman et al. [5]. A challenge is to identify the near-peer mentors, people who have joined the project in a certain time frame or reached a certain milestone (e.g., gotten the first Pull Request accepted). Another challenge, is providing meaningful incentives to mentors, for example, recognition or achievement points in the community.

Transferring knowledge across projects in an OSS ecosystem. Having (available) tasks from multiple projects that share the same technical platform, infrastructure, or build on the stack can serve dual purposes. First, it will provide a steady set of tasks at different levels of complexity. Second, it will allow newcomers to learn from one project and apply to another. Moreover, it will also allow the creation of a more diverse set of peer-mentor networks. However, the challenge lies in: (1) identifying the projects that constitute an ecosystem (e.g., projects to which developers contribute in parallel, or projects that share technical dependencies or form part of a stack); and (2) the feasibility of learning from performing tasks in one project and applying that knowledge to another project, within the same ecosystem.

Using labeled tasks to identify microtasks to be crowdsourced: Our approach may be extended to a crowd development model. One of the challenges of crowdsourcing software development is to break the tasks (or identify) microtasks. Microtasks are described by LaToza et al. [10] as short, self-descriptive and modular tasks, that allow immediate contributions, without deep knowledge of the project. Since our approach can help identify easy issues, it can be used to point to microtasks that can be crowdsourced. The main challenges here are the proper identification of microtasks and the ability to create coordination and incentive models for the crowd workers.

5. BugExchange

5.1 Preliminary Work

We have already started our work on two fronts. The first is to see if project newcomers can label the skills required for a task, and the second is the development of the infrastructure on which our ideas will be built.

5.1.1 Task Labeling

Typically, OSS developers need to find task that they can implement. That is, they have to go through a list of available tasks and determine which one is the most suited for them [3]. However, their lack of experience in the project is often a hurdle [17].

We performed a study to investigate how well newcomers to a project could *label the tasks* with the skills required to complete the task. To do so, we employed nine crowd workers from ODesk to analyze a task's context (description, discussion, and source code), and propose the skills they believed were relevant to solve the task. The crowd workers had some Java experience, but were inexperienced (like newcomers) to the project. We found that it is possible to obtain most of the relevant skills by using a voting mechanism to filter the output from multiple workers [11]. Our results show that participants took about 30 minutes to perform the labeling, which included reviewing the task and its associated codebase. The minimum number of workers required (per task) to get the best results was four. By using an agreement threshold of 25%, we obtained a 0.67 recall (missing few relevant skills) and 0.76 precision (adding few extraneous skills).

5.1.2 FLOSScoach

Based on the barriers model proposed in our previous study [17], we built FLOSScoach, a portal to support the first steps of newcomers to OSS projects [16]. The portal has been structured to reflect the categories identified in the barriers model. Each category was mapped onto a portal section that contains information and strategies aimed at supporting newcomers in overcoming the identified barriers. In the portal, newcomers find information on the skills needed to contribute to a project, a step-by-step contribution flow, the location of features (such as source code repository, issue tracker and mailing list), and tips on how to interact with the community and how to submit a patch.

Our preliminary study has shown that FLOSScoach helps newcomers, by guiding them in their first steps, and increasing their confidence in their ability to contribute to a project [16]. When we compared students' performance with and without FLOSScoach, we found a significant drop in terms of self-efficacy among students in the control group (not using FLOSScoach), while the self-efficacy of students using the tool remained at a high level. In addition, by analyzing diaries written during the contribution process, we found evidence that FLOSScoach made newcomers feel oriented and more comfortable with the process, while those who did not have access to FLOSScoach, repeatedly reported uncertainty and doubt on how to proceed.

5.2 Evaluation Plans

The aforementioned challenges will unfold in several research lines. Each one will have its individual evaluation plan. However, it is important to evaluate how they fit together. Given the breadth of our approach, there are several research questions that need answers: How effective are newcomers in labeling the required skills of a task? How accurate are the task complexity measurements? Are the identified information needs enough for a newcomer to accomplish a task? Does the provided documentation

support task resolution? Are task recommendations appropriate to the newcomer skills, motivations, and availability? Do the hints and mentoring provided by the peers help newcomers in their progress? What are newcomers learning while accomplishing tasks and progressing through the different levels of difficulty? How do they move across different levels of complexity of tasks? How much knowledge is being transferred across projects?

To evaluate such a complex scenario in which learning, motivation, and inter-related activities play a role, it is necessary to conduct a series of long-term, in-depth studies to evaluate how newcomers follow our approach.

We plan to implement the approach into FLOSScoach. Similarly to our prior work [16], we will evaluate the proposed approach with software engineering students from multiple universities, where they have to contribute to OSS as part of their coursework. These students are exemplar newcomers, as they are new to the OSS ecosystem, to the specific project, and to software development in general. We will use diary studies to monitor the students' progress and to collect their impressions and tool usage. The students will be required to log their task activities, any issues they encountered, and everything else they did while working on the tasks in a shared document. This kind of study enables access to everyday behavior in a relatively unobtrusive manner, which affords access to the experience's immediacy, and also provides accounts of phenomena over time. While students will be free to work on a project of their choice and at a time in their own discretion, we will ensure that the project selection is restricted within a pre-determined ecosystem.

We will complement the diary studies with specific questionnaires, such as pre- and post-study self-efficacy and Technology Acceptance Model (TAM). Self-efficacy is a measure of the confidence in the participants' perceived ability to perform a task, which can impact one's actual ability to complete a task [1]. TAM is a model that assesses user perceptions about a technology's usefulness, usability, and future use. We will also interview instructors, project members, and run controlled user studies. Additionally, we will deploy our approach in multiple projects and observe its use in the "wild". We will conduct surveys and interviews of newcomers in these projects to get feedback.

6. CONCLUSION

Open Source Software (OSS) projects have become prominent and support a large number of today's society activities, becoming an important economic driving force. Supporting newcomers to the OSS projects is crucial for this whole ecosystem. Newcomers do not know how to start or choose inappropriate tasks and end up giving up [18]. We claim that if we can adequately support the curation of tasks as a newcomer activity, many benefits will follow: newcomers will gain understanding about the project and the technologies and concepts involved, projects will have issue trackers more organized, and newcomers will have more information to support their decision on where to start. However, many challenges are involved, such as providing adequate means to newcomers evaluate the tasks, processing documentation and source code, creating recommendation systems to aid the process, proposing collective validation strategies for the information provided, etc. Different research lines are necessary to overcome these challenges. We will leverage our previous work, aggregating the results of the individual research lines, testing how well we are supporting newcomers, and the projects in their sustainability. We conjecture that BugExchange may reduce newcomer dropouts and foster more casual contributors [14] into the projects.

7. REFERENCES

[1] Bandura, A. 1986. Social foundations of thought and action: a social cognitive theory. Prentice-Hall.

[2] Cubranic, D. et al. 2005. Hipikat: a project memory for software development. IEEE Transactions on Software Engineering. 31, 6 (Jun. 2005), 446–465.

[3] Ducheneaut, N. 2005. Socialization in an Open Source Software Community: A Socio-Technical Analysis. Computer Supported Cooperative Work. 14, 4, 323–368.

[4] Fagerholm, F. et al. 2014. Onboarding in Open Source Projects. IEEE Software. 31, 6 (Nov. 2014), 54–61.

[5] Glassman, E.L. et al. 2016. Learnersourcing Personalized Hints. *19th ACM CSCW*, 1626–1636.

[6] Hars, A. and Ou, S. 2001. Working for free? Motivations of participating in open source projects. HICSS 2001, 1–9.

[7] Jensen, C. et al. 2011. Joining Free/Open Source Software Communities: An Analysis of Newbies' First Interactions on Project Mailing Lists. HICSS 2011, 1–10.

[8] Jergensen, C. et al. 2011. The Onion Patch: Migration in Open Source Ecosystems. ESEC/FSE 2011, 70–80.

[9] Krogh, G. von and Hippel, E. von 2003. Editorial: Special issue on open source software development. Research Policy. 32, 7 (Jul. 2003), 1149–1157.

[10] LaToza, T.D. et al. 2013. Crowd development. CHASE 2013, 85–88.

[11] Leaño, R. et al. 2016. Labeling relevant skills in tasks: can the crowd help? IEEE VL/HCC 2016.

[12] Park, Y. and Jensen, C. 2009. Beyond pretty pictures: Examining the benefits of code visualization for open source newcomers. 5th IEEE VISSOFT, 3-10.

[13] Parnin, C. and Treude, C. 2011. Measuring API Documentation on the Web. 2nd International Workshop on Web 2.0 for Software Engineering, 25–30.

[14] Pinto, G. et al. 2016. More Common Than You Think: An In-Depth Study of Casual Contributors. SANER 2016.

[15] Qureshi, I. and Fang, Y. 2011. Socialization in Open Source Software Projects: A Growth Mixture Modeling Approach. Organizational Research Methods. 14, 1, 208–238.

[16] Steinmacher, I. et al. 2016. Overcoming Open Source Project Entry Barriers with a Portal for Newcomers. ICSE 2016.

[17] Steinmacher, I. et al. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. 18th ACM CSCW 2015, 1–13.

[18] Steinmacher, I. et al. 2015. Understanding and Supporting the Choice of an Appropriate Task to Start With In Open Source Software Communities. HICSS 2015, 1–10.

[19] Steinmacher, I. et al. 2013. Why do newcomers abandon open source software projects? CHASE 2013, 25–32.

[20] Storey, M.-A. et al. 2010. The Impact of Social Media on Software Engineering Practices and Tools. FSE/SDP Workshop on Future of Software Engineering Research, 359–364.

[21] Wang, J. and Sarma, A. 2011. Which bug should I fix: helping new developers onboard a new project. CHASE 2011, 76–79.

[22] Zhou, M. and Mockus, A. 2010. Developer fluency: Achieving true mastery in software projects. FSE 2010, 137–146.

[23] Zhou, M. and Mockus, A. 2011. Does the initial environment impact the future of developers. ICSE 2011, 271–280.

[24] Zhou, M. and Mockus, A. 2015. Who Will Stay in the FLOSS Community? Modelling Participant's Initial Behaviour. IEEE Tran on Soft Eng. 41, 1, 82–99